

CMOS 8-Bit Single-Chip Microcontroller

# **MSM80C31** **MSM80C51**

## **USER'S MANUAL**

**PROMILECT**

Reg. No. 64 1688 07

P.O. BOX 1194 RANDBURG 2125

M.F. KENT HOUSE DOVER STREET RANDBURG TVL 2194  
TEL (0111) 789 1400/2 TELEX 4-20462 FAX (0111) 7870263



quantum electronics

Box 391262

Bramley

2018

# **OKI**

**SEMICONDUCTOR**

First Edition December 1985

# ***MSM80C31*** ***MSM80C51***

---

## ***USER'S MANUAL***

quantum electronics  
Box 391262  
Bramley  
2018





# CONTENTS

<b>PREFACE</b>	i
<b>1. INTRODUCTION</b>	2
1.1 MSM80C31/MSM80C51 Outline	2
1.2 MSM80C31/MSM80C51 Features	3
<b>2. MSM80C31/MSM80C51 SYSTEM CONFIGURATION</b>	6
2.1 MSM80C31/MSM80C51 Logic Symbols	6
2.2 MSM80C31/MSM80C51 Pin Layout	7
2.2.1 External dimensions	8
2.3 MSM80C31 Internal Block Diagram	9
2.4 MSM80C51 Internal Block Diagram	10
2.5 Timing and Control	11
2.5.1 Outline of MSM80C31/MSM80C51 timing	11
2.5.2 Major Synchronizing signals	13
2.5.3 MSM80C31 fundamental operation time chart	14
2.5.4 MSM80C51 fundamental operation time chart	17
2.6 Instruction Register (IR) and Instruction Decoder (PLA)	20
2.7 Arithmetic Operation Section	21
2.8 Program Counter	22
2.9 Program Memory and External Data Memory	23
2.9.1 MSM80C31/MSM80C51 program area and external ROM connections	23
2.9.2 Procedures and circuit connections used when external data memory (RAM) is read/written by data pointer	25
2.9.3 Procedures and circuit connections used when external data memory (RAM) is read/written by register R0 and R1	28
<b>3. MSM80C31/MSM80C51 CONTROL</b>	32
3.1 Oscillators: XTAL 1·2	32
3.2 CPU Resetting	34
3.2.1 Outline	34
3.2.2 Reset schmitt trigger circuit	39
3.2.3 CPU reset internal status	40
3.3 $\overline{EA}$ (CPU Memory Separate)	41
3.3.1 Outline	41
<b>4. MSM80C31/MSM80C51 INTERNAL SPECIFICATIONS</b>	44
4.1 Internal Data Memory (RAM) and Special Function Registers (SFR)	44
4.1.1 Outline	44
4.2 Internal Data Memory (RAM)	46
4.2.1 Internal data memory (RAM)	46
4.2.2 Internal data memory registers R0 thru R7	48
4.2.3 Stack	49

4.3 Internal Data Memory (RAM) Operating Procedures .....	50
4.3.1 Internal data memory indirect addressing .....	50
4.3.2 Internal data memory register R0 thru R7 designation .....	51
4.3.3 Internal data memory 1-bit data designation .....	52
4.4 Special Function Registers (TCON, SCON, ... ACC, B) .....	54
4.4.1 Outline .....	54
4.4.2 Special function registers .....	56
4.4.2.1 Timer mode register (TMOD) .....	56
4.4.2.2 Power control register (PCON) .....	57
4.4.2.3 Timer control register (TCON) .....	58
4.4.2.4 Serial port control register (SCON) .....	59
4.4.2.5 Interrupt enable register (IE) .....	60
4.4.2.6 Interrupt priority register (IP) .....	61
4.4.2.7 Program status word register (PSW) .....	62
4.5 Timer/Counters 0 and 1 .....	63
4.5.1 Outline .....	63
4.5.2 Timer/counter 0 and 1 counting control .....	63
4.5.3 Timer/counter 0 and 1 clock designation .....	65
4.5.3.1 External clock detector circuit for timer/counters 0 and 1 .....	65
4.5.4 Counting control of timer/counters 0 and 1 by $\overline{\text{INT}}$ pin .....	67
4.5.5 Timer/counters 0/1 and timer modes .....	69
4.5.5.1 Outline .....	69
4.5.5.2 Mode 0 .....	69
4.5.5.3 Mode 1 .....	71
4.5.5.4 Mode 2 .....	73
4.5.5.5 Mode 3 .....	75
4.5.6 Timer/counter carry signal detector circuit .....	76
4.6 Serial Port .....	77
4.6.1 Outline .....	77
4.6.2 Special function registers for serial port .....	78
4.6.2.1 SCON (serial port control register) .....	78
4.6.2.2 SBUF (serial port buffer register) .....	79
4.6.2.3 SMOD (double baud rate bit) .....	79
4.6.3 Operating modes .....	80
4.6.3.1 Mode 0 .....	80
4.6.3.1.1 Outline .....	80
4.6.3.1.2 Mode 0 baud rate .....	80
4.6.3.1.3 Mode 0 output operation .....	80
4.6.3.1.4 Mode 0 input operation .....	80
4.6.3.2 Mode 1 .....	84
4.6.3.2.1 Outline .....	84
4.6.3.2.2 Mode 1 baud rate .....	84
4.6.3.2.3 Mode 1 transmit operation .....	84
4.6.3.2.4 Mode 1 receive operation .....	87

4.6.3.3 Mode 2 .....	88
4.6.3.3.1 Outline .....	88
4.6.3.3.2 Mode 2 baud rate .....	88
4.6.3.3.3 Mode 2 transmit operation .....	88
4.6.3.3.4 Mode 2 receive operation .....	88
4.6.3.4 Mode 3 .....	92
4.6.3.4.1 Outline .....	92
4.6.3.4.2 Mode 3 baud rate .....	92
4.6.3.4.3 Mode 3 transmit operation .....	92
4.6.3.4.4 Mode 3 receive operation .....	95
4.6.4 Serial port application examples .....	96
4.6.4.1 I/O extension .....	96
4.6.4.2 Multi-processor systems .....	100
4.7 Interrupts .....	102
4.7.1 Outline .....	102
4.7.2 Interrupt enable register (IE) .....	104
4.7.3 Interrupt priority register (IP) .....	105
4.7.3.1 Priority interrupt process flow .....	106
4.7.3.2 Interrupt priority when priority register (IP) contents are all "0" .....	107
4.7.4 Detection of external interrupt signals $\overline{INT0}$ and $\overline{INT1}$ .....	108
4.7.4.1 Outline of $\overline{INT}$ signal detection .....	108
4.7.4.2 External interrupt signal 0 and 1 level detection .....	108
4.7.4.3 External interrupt signal 0 and 1 trigger detection .....	110
4.7.5 MSM80C31/MSM80C51 interrupt response time charts .....	111
4.7.5.1 Interrupt response time chart when interrupt conditions are satisfied during execution of ordinary instruction .....	111
4.7.5.2 Interrupt response time chart when interrupt conditions are satisfied during execution of IE or IP register operation instruction .....	112
4.7.5.3 Interrupt response time chart when an ordinary instruction is executed after temporarily returning to the main routine from continuous interrupt processing .....	113
4.7.5.4 Interrupt response time chart when an IE or IP manipulating instruction is executed after temporarily returning to the main routine from continuous interrupt processing .....	114
4.8 CPU "Power Down" .....	115
4.8.1 Outline .....	115
4.8.2 Idle mode (IDLE) setting .....	115
4.8.3 Power down mode (PD) setting .....	119
4.9 CPU Power Down (IDLE and PD) Cancellation (CPU Activation) .....	123
4.9.1 Outline .....	123
4.9.2 Cancellation by CPU resetting .....	123
4.9.3 Reactivation of CPU by an interrupt .....	128

<b>5. INPUT/OUTPUT PORTS</b>	132
5.1 Outline	132
5.2 Port 0	132
5.3 Port 1	134
5.4 Port 2	137
5.5 Port 3	139
5.6 Port Output Timing	141
5.7 Port Data Manipulating Instructions	143
<b>6. MSM80C31/MSM80C51 ELECTRICAL CHARACTERISTICS</b>	146
6.1 Absolute Maximum Ratings	146
6.2 Operation Ranges	146
6.3 DC Characteristics	147
6.4 External Program Memory Access AC Characteristics	150
6.5 External Data Memory Access AC Characteristics	152
6.6 Serial Port (I/O Extension Mode) AC Characteristics	154
6.7 AC Characteristics Measuring Conditions	156
<b>7. DESCRIPTIONS OF INSTRUCTIONS</b>	158
7.1 Outline	158
7.2 Description of Instruction Symbols	159
7.3 List of Instructions	160
7.4 Simplified Description of Instructions	161
7.5 Detailed Description of MSM80C31/MSM80C51 Instructions	173
<b>8. MSM80C51VS PIGGY BACK</b>	302

## PREFACE

This manual describes the hardware and instructions for the

MSM80C51RS,  
MSM80C51GS,  
MSM80C51JS,  
MSM80C31RS,  
MSM80C31GS,  
MSM80C31JS, and  
MSM80C51VS

CMOS 8-bit microcontrollers.

RS .....	40-pin plastic DIP package
GS .....	44-pin plastic flat package
JS .....	44-pin PLCC package
VS .....	40-pin ceramic DIP package (Piggy-back type)

These microcontrollers are represented in the manual by MSM80C31/  
MSM80C51.



# 1. INTRODUCTION

used in the manufacture of these microcontrollers.

MSM80C31 is the same as MSM80C31 but without the internal program memory (ROM). MSM80C31VS is a device whose internal program memory is replaced by external EPROM which is piggy backed to the CPU.

While the MSM80C31 microcontroller features an additional 4096-word  $\times$  8-bit program memory (ROM) mounted on a single chip, both the MSM80C31 and MSM80C31VS include a 128-word  $\times$  8-bit data memory (RAM), 32 input/output ports, two 16-bit I/O timer/counters, five interrupts, a serial I/O, an 8-bit parallel processing circuit, and a clock generator integrated on a single chip.

The internal operation in these CPUs is based on an instruction code address method for greater efficiency. The operations are specified in the instruction code (OP) section, and the objective registers are specified by part of the instruction code (OP) and the second or third byte following the instruction code. A feature of this method is the ability to enable several operations by only changing the operation register designation in a single instruction code.

The inclusion of 8-bit multiplication and division instructions further increases the processing capacity of these CPUs.

In addition to expansion of the bit processing area, a comprehensive range of bit processing instructions has also been included. Processing operations include logical processing of carry flag and specified bit within each register, transfer between carry flag and specified bit in certain register, setting, resetting, and complement of specified bit in each register, and execution of various bit tests within a wide area.

To make a relative jump after the execution of a branch instruction (bit test instruction), jumps can be made within a wide address range between  $-128$  to  $+127$  relative to the address of the instruction. There is no page field restriction as in other systems. (Jumps are made within pages in other systems.)

The contents of specified registers can be saved in stack by using the PUSH instruction, and the saved contents can be returned from stack to specified register by the POP instruction. Allocation of absolute interrupt priority to any interrupt makes interrupt processing all the easier.

Employing the low-power consumption feature of C-MOS device, these CPUs are designed to operate in "CPU power down" mode. In idle mode (IDLE) the IDLE bit in the power control register (PCON) is set to "1" to halt CPU operations while continuing XTAL1 and XTAL2 operations. In the power down mode (PDM) the PD bit in the power control register (PCON) is set to "1" to halt CPU operations as well as the XTAL1 and XTAL2 operations. The CPU power down mode can be cancelled by a reset or an interrupt.

Two built-in 16-bit timer/counters capable of a wide range of operational mode enable the CPU to be used in many different ways.

UART based serial interface can be executed at any baud rate by clock signals from timer/counter 1.



# 1. INTRODUCTION

## 1.1 MSM80C31/MSM80C51 Outline

MSM80C31/MSM80C51 are single-chip 8-bit microcontrollers featuring high performance and low power consumption. 2  $\mu$ m silicon gate processing technology has been used in the manufacture of these microcontrollers.

MSM80C31 is the same as MSM80C51 but without the internal program memory (ROM). MSM80C51VS is a device whose internal program memory is replaced by external EPROM which is piggy backed to the CPU.

While the MSM80C51 microcontroller features an additional 4096-word  $\times$  8-bit program memory (ROM) mounted on a single chip, both the MSM80C31 and MSM80C51 include a 128-word  $\times$  8-bit data memory (RAM), 32 input/output ports, two 16-bit R/W timer/counters, five interrupts, a serial I/O, an 8-bit parallel processing circuit, and a clock generator integrated on a single chip.

The internal operation in these CPUs is based on an instruction code address method for greater efficiency. The operations are specified in the instruction code (OP) section, and the objective registers are specified by part of the instruction code (OP) and the second or third byte following the instruction code. A feature of this method is the ability to enable several operations by only changing the operation register designation in a single instruction code.

The inclusion of 8-bit multiplication and division instructions further increases the processing capacity of these CPUs.

In addition to expansion of the bit processing area, a comprehensive range of bit processing instructions has also been included. Processing operations include logical processing of carry flag and specified bit within each register, transfer between carry flag and specified bit in certain register, setting, resetting, and complement of specified bit in each register, and execution of various bit tests within a wide area.

To make a relative jump after the execution of a branch instruction (bit test instruction), jumps can be made within a wide address range between  $-128 \sim +127$  relative to the address of the instruction. There is no page field restriction as in previous systems. (Jumps are made within pages in other systems.)

The contents of specified registers can be saved in stack by using the PUSH instruction, and the saved contents can be returned from stack to specified register by the POP instruction. Allocation of absolute interrupt priority to any interrupt makes interrupt processing all the easier.

Employing the low-power consumption feature of C-MOS devices, these CPUs are designed to operate in "CPU power down" modes. In idle mode (IDLE) the IDL bit in the power control register (PCON) is set to "1" to halt CPU operations while continuing XTAL1 and XTAL2 operations. In the power down mode (PD) the PD bit in the power control register (PCON) is set to "1" to halt CPU operations as well as the XTAL1 and XTAL2 operations. The CPU power down mode can be cancelled by a reset or an interrupt.

Two built-in 16-bit timer/counters capable of a wide range of operational mode enable the CPUs to be used in many different ways.

UART based serial interface can be executed at any baud rate by clock signals from timer/counter 1.

The comprehensive range of functions on this CPU gives a highly integrated high performance solution in a very short time.

## 1.2 MSM80C31/MSM80C51 Features

- Si gate C-MOS
- Program memory (ROM)  
4096 words  $\times$  8 bits (MSM80C51)
- Data memory (RAM)  
128 words  $\times$  8 bits
- Stack  
Freely usable 128-word  $\times$  8-bit data memory area
- Four sets of working registers (R0 thru R7  $\times$  4)
- Five types of interrupts (with priority)
  - (1) Two external interrupts
  - (2) Two timer interrupts
  - (3) Serial port interrupt
- External data memory (RAM)  
Up to 64K bytes
- External program memory  
Up to 64K bytes
- Four Input/Output ports (8-bit  $\times$  4)
- CPU power down function
  - (1) Idle mode: CPU stopped while XTAL1  $\cdot$  2 operations continued.  
(Software setting)
  - (2) Power Down mode: CPU and XTAL1  $\cdot$  2 all stopped. (Software setting)
- Two 16-bit timers/counters
  - Timer mode
    - 1) 8-bit timer with 5-bit prescaler
    - 2) 16-bit timer
    - 3) 8-bit timer with 8-bit auto reloader
    - 4) 8-bit separate timer
- Serial port (UART mode operation)  
Multiple baud rates
- Direct initialization of input/output ports by RESET signal even if XTAL1  $\cdot$  2 have been stopped ("1" data set)
- Instruction execution cycle 1  $\mu$ sec. @ 12 MHz  
( $V_{cc} = +5 [V] \pm 20\%$ )
- Wide-range of clock operation (DC to 12 MHz)  
(XTAL1  $\cdot$  2 operation frequency dependent on  $V_{cc}$ )
- Wide-range of operating power supply
  - (1) When operating:  $V_{cc} = +2.5$  to  $+6 V$
  - (2) CPU internal data hold voltage (XTAL1  $\cdot$  2 both stopped)  
 $V_{cc} = +2$  to  $+6 V$
- Wide-range of operating temperature ( $-40 \sim +85^{\circ}C$ )
- High noise margin (with Schmitt trigger input for each I/O)
- 40-pin plastic DIP / 44-pin plastic flat package/44-pin PLCC
- Software compatibility with Intel 80C31/80C51



# 2. MSM80C31/ MSM80C51<sup>2</sup> SYSTEM CONFIGURATION

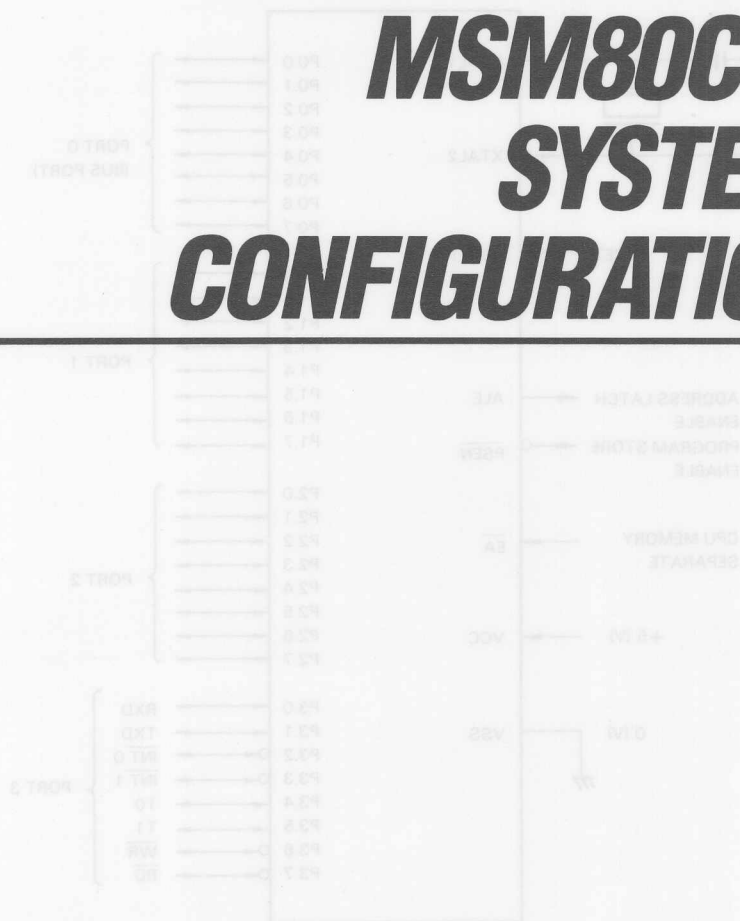


Figure 2-1 MSM80C31/MSM80C51 logic symbols

## 2. MSM80C31/MSM80C51 SYSTEM CONFIGURATION

### 2.1 MSM80C31/MSM80C51 Logic Symbols

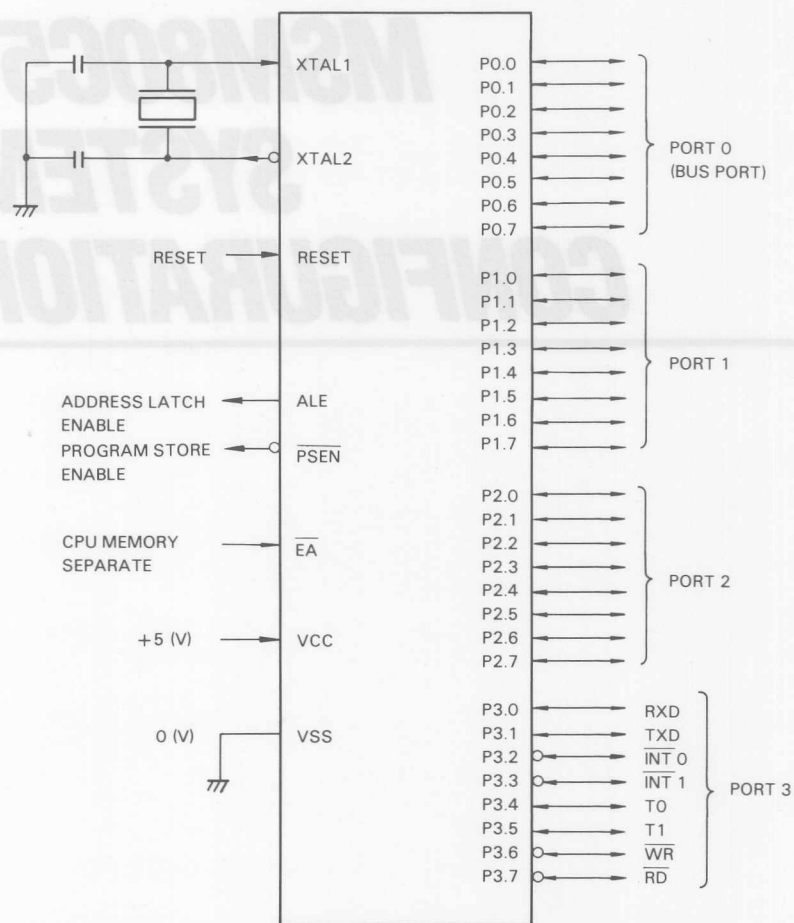
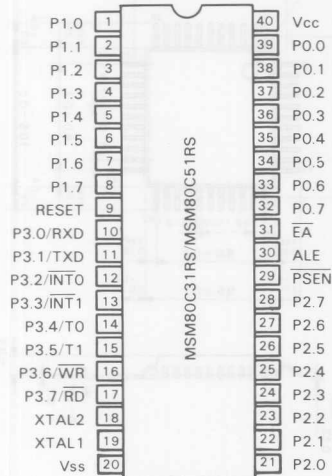


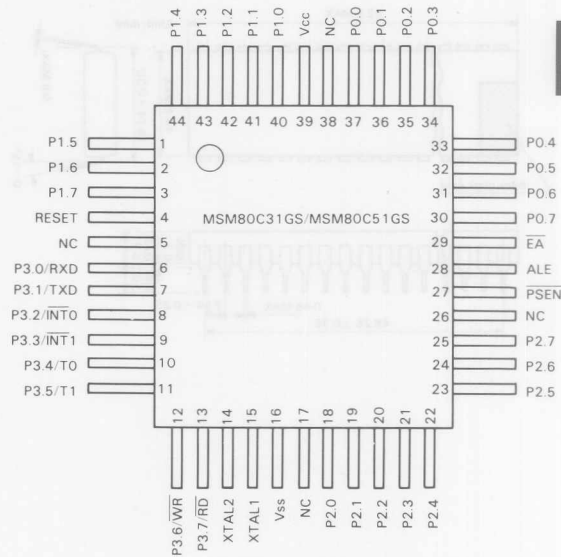
Figure 2-1 MSM80C31/MSM80C51 logic symbols

## 2.2 MSM80C31/MSM80C51 Pin Layout

MSM80C31RS/MSM80C51RS  
(Top View) 40 Lead Plastic DIP



MSM80C31GS/MSM80C51GS  
(Top View) 44 Lead Plastic Flat Package



MSM80C31JS/MSM80C51JS  
(Top View) 44 Lead Plastic Leaded Chip Carrier

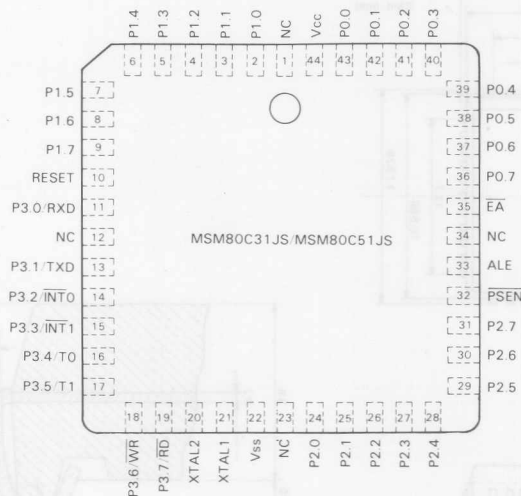
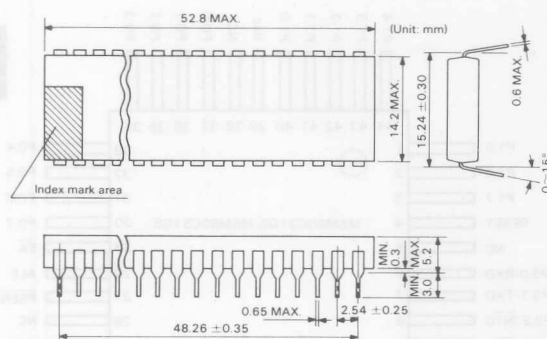


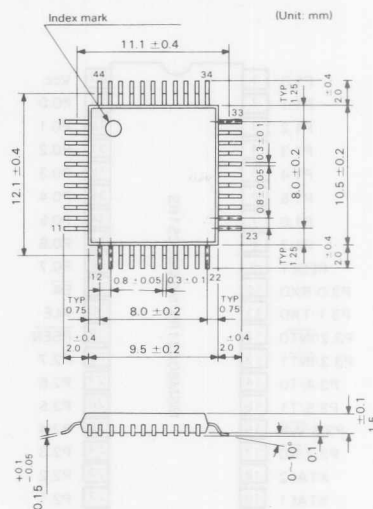
Figure 2-2 MSM80C31/MSM80C51 pin layout

## 2.2.1 External dimensions

MSM80C31RS/MSM80C51RS  
[40 Lead Plastic DIP]

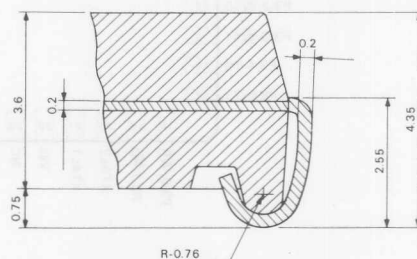
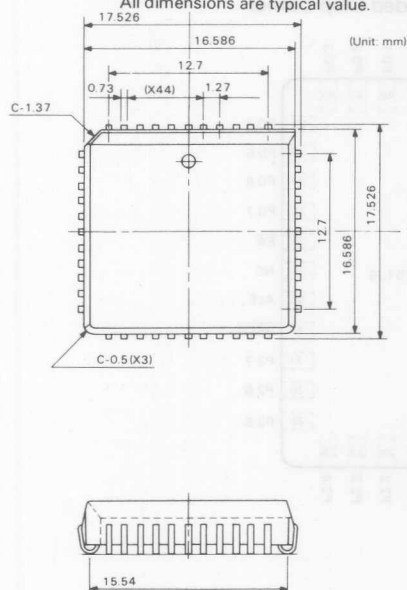


MSM80C31GS/MSM80C51GS  
[44 Lead Plastic Flat Package]



MSM80C31JS/MSM80C51JS  
[44 Lead Plastic Leaded Chip Carrier]

All dimensions are typical value.



## 2.3 MSM80C31 Internal Block Diagram

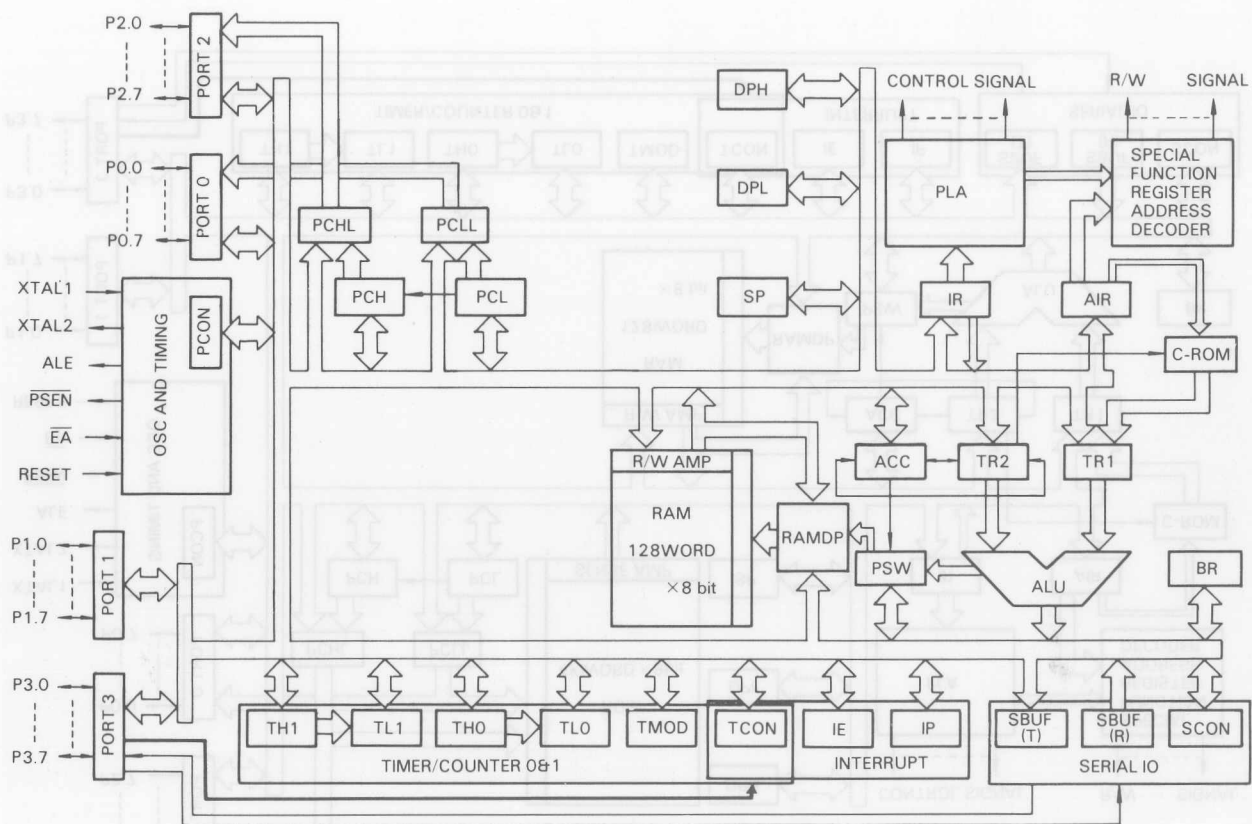


Figure 2-3 MSM80C31 internal block diagram



## 2.4 MSM80C51 Internal Block Diagram

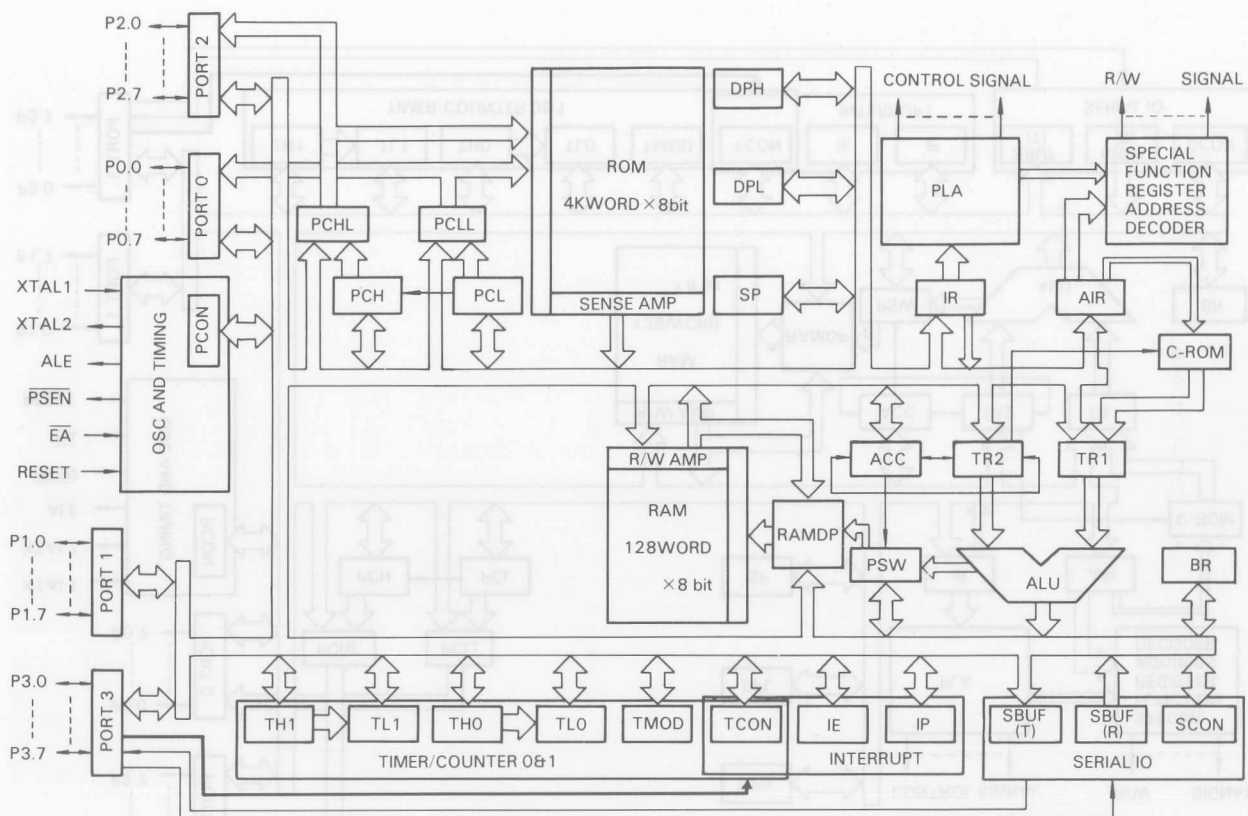


Figure 2-4 MSM80C51 internal block diagram

## 2.5 Timing and Control

### 2.5.1 Outline of MSM80C31/MSM80C51 timing

The MSM80C31/MSM80C51 is equipped with a built-in inverter (see Figure 2-5) for use in the generation of clock pulses by external crystal or ceramic resonator. These clock pulses are passed to the timing counter and control circuits where the basic timing and control signals required for internal control are generated.

The basic timing consists of S1 thru S6 (see Figure 2-6) where each timing cycle is based on the  $2 \text{ XTAL1} \cdot 2$  fundamental clock pulses. The interval from S1 thru S6 forms a single machine cycle with a total of 12 clock pulses.

1-byte 1-machine cycle and 2-byte 1-machine cycle instructions are fetched by instruction register during M1 · S1 cycle, decoded during M1 · S2 cycle, and executed during M1 · S3 thru M1 · S6 cycle. The second byte is fetched during M1 · S4 cycle.

1-byte 2-machine cycles, 2-byte 2-machine cycles, and 3-byte 2-machine cycles instructions are also fetched during M1 · S1, decoded during M1 · S2, and executed during M1 · S3 thru M2 · S6. The second byte and third byte is fetched during M1 · S4, M2 · S1, or M2 · S4. The number of clocks used is 24. 1-byte 4-machine cycles instructions are involved in multiplication and division operations where 48 clocks are used.

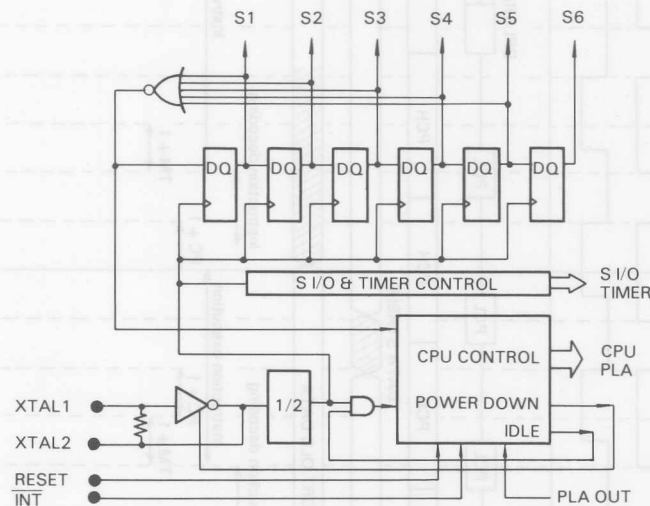


Figure 2-5 Oscillator, timing counter, and control stage block diagram

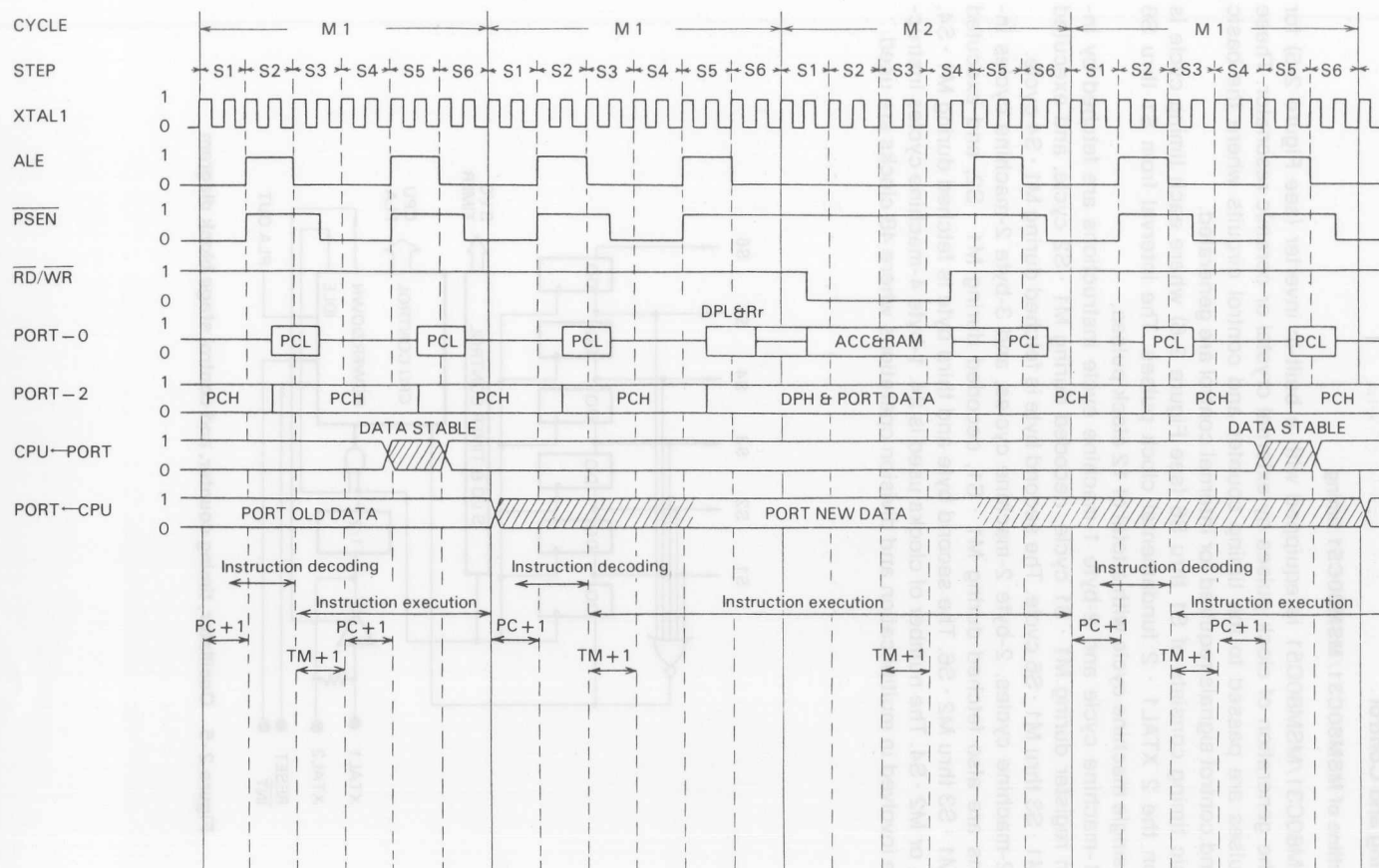


Figure 2-6 MSM80C31/MSM80C51 fundamental timing

## 2.5.2 Major synchronizing signals

## (1) ALE (Address Latch Enable)

The ALE signal is used as clock signal where the output address signals 0 thru 7 from CPU port 0 can be latched externally when external program memory or external data memory (RAM) is used.

Although two ALE signal outputs are obtained in a single machine cycle during normal operations, no output is obtained during output of the  $\overline{RD}/\overline{WR}$  signal when an external memory instruction (MOVX ..... ) is executed.

(2)  $\overline{PSEN}$  (Program Store Enable)

The  $\overline{PSEN}$  output signal is generated during execution of an external program. This output is obtained during the fetch of instructions and data.

The  $\overline{PSEN}$  signal is a low active signal, and external program data is enabled when this signal is active.

Although two  $\overline{PSEN}$  signal outputs are obtained in a single machine cycle during normal operations, no output is obtained during output of the  $\overline{RD}/\overline{WR}$  signal when an external memory instruction (MOVX ..... ) is executed.

(3)  $\overline{WR}$  (Write Strobe)

The  $\overline{WR}$  output signal is obtained when an external memory instruction (MOVX @Rr, A or MOVX @DPTR, A) is executed.

CPU port 0 output data is written in the external RAM when the  $\overline{WR}$  signal is low.

(4)  $\overline{RD}$  (Read Strobe)

The  $\overline{RD}$  output signal is obtained when an external memory instruction (MOVX A, @Rr or MOVX A, @DPTR) is executed.

The external RAM is enabled and output data is passed to CPU port 0 when the  $\overline{RD}$  signal is low.

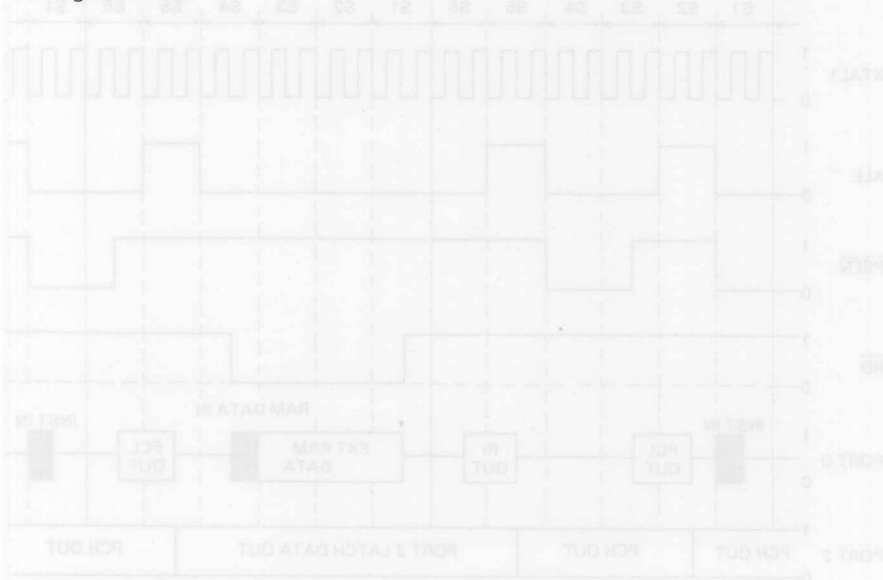


Figure 2-8 MSM80C31 MOVX A, @DPTR execution

## 2.5.3 MSM80C31 fundamental operation time chart

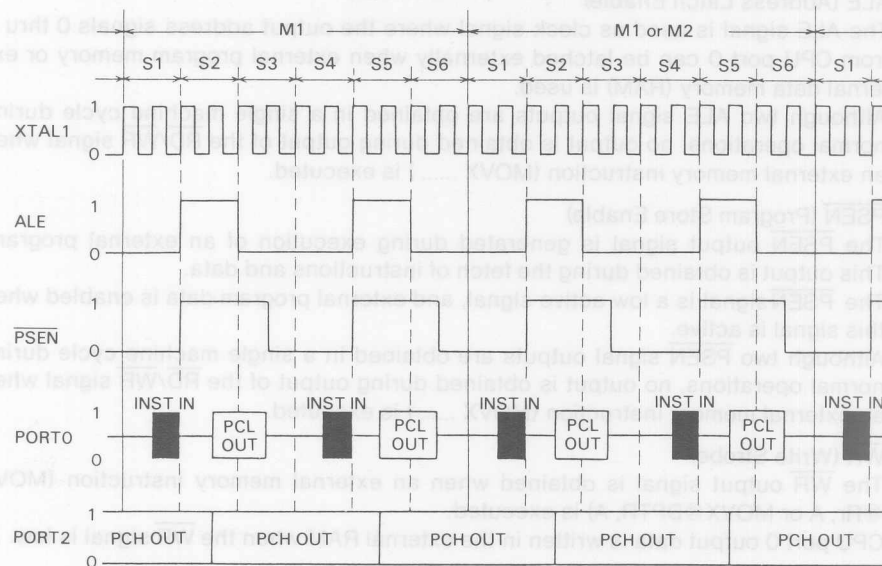


Figure 2-7 MSM80C31 external program memory read cycle timing chart

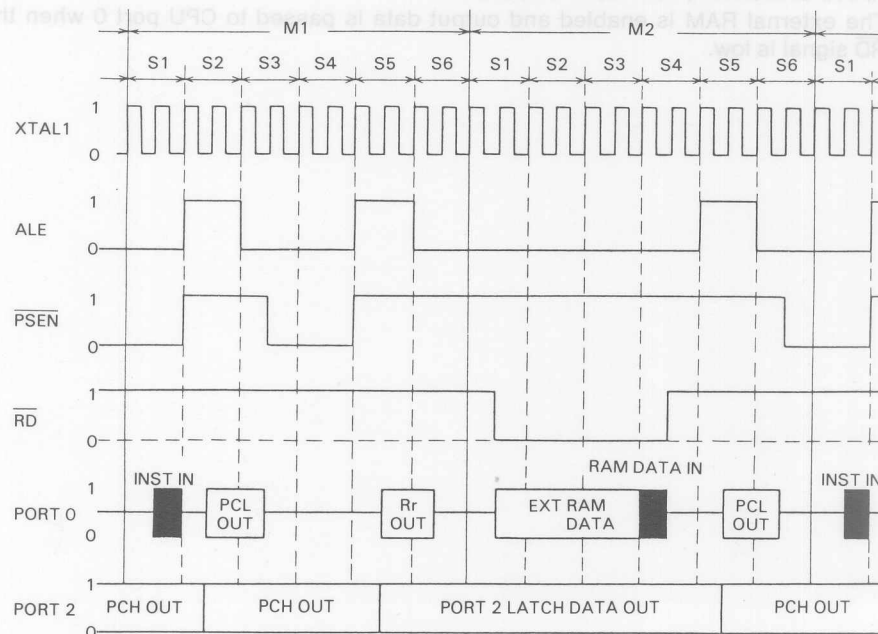


Figure 2-8 MSM80C31 MOVX A, @Rr execution

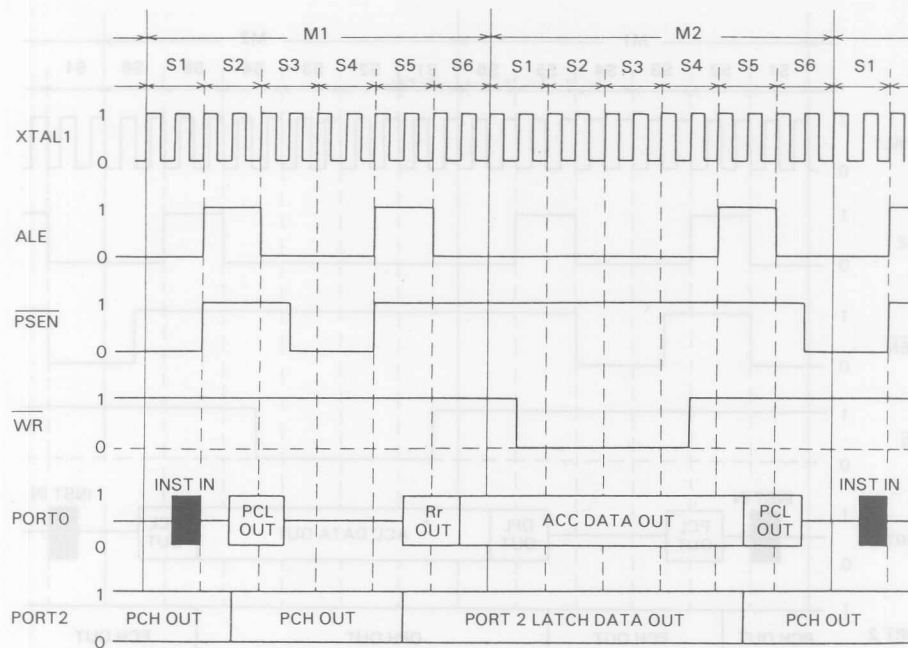


Figure 2-9 MSM80C31 MOVX @Rr, A execution

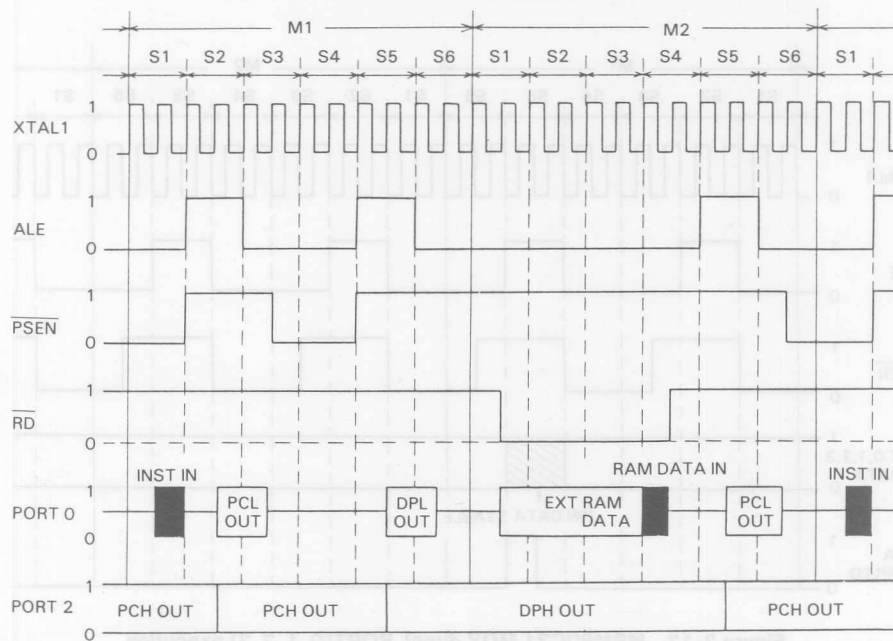


Figure 2-10 MSM80C31 MOVX A, @DPTR execution

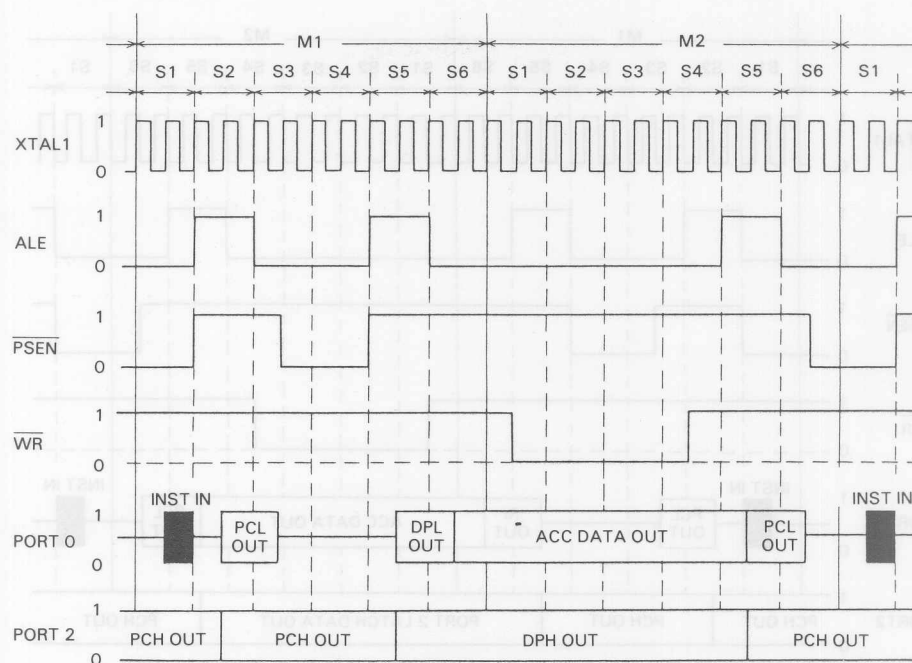


Figure 2-11 MSM80C31 MOVX @DPTR, A execution

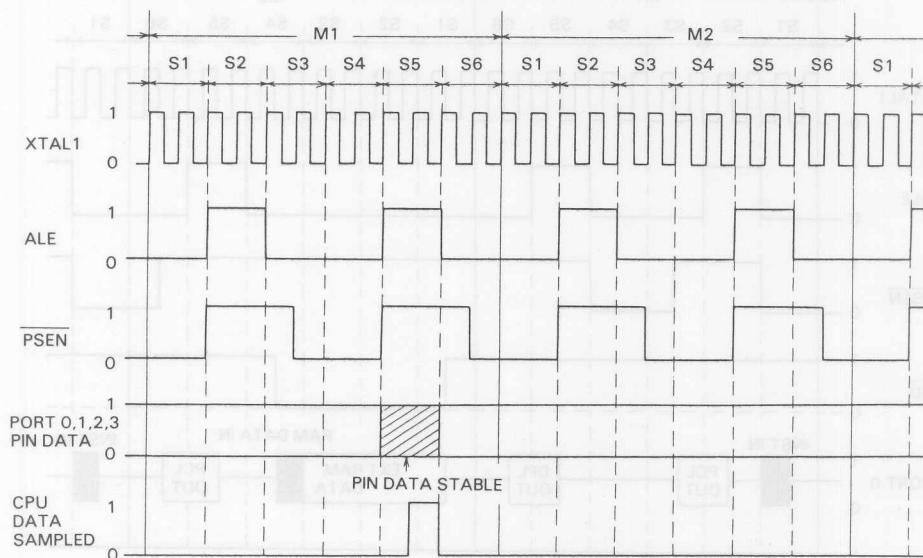


Figure 2-12 MSM80C31 MOV direct, PORT[0, 1, 2, 3] execution

# 2.5.4 MSM80C51 fundamental operation time chart

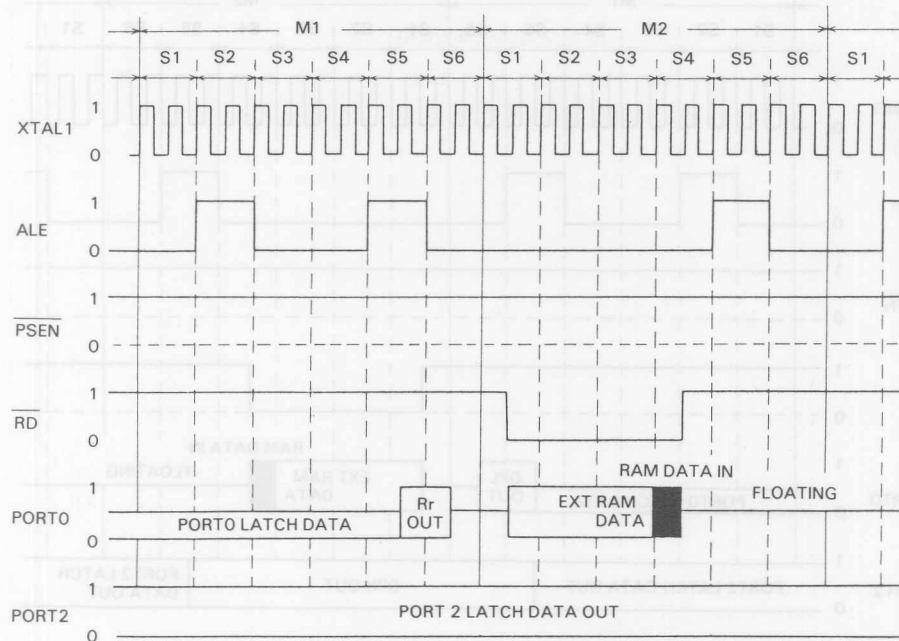


Figure 2-13 MSM80C51 MOVX A, @Rr execution

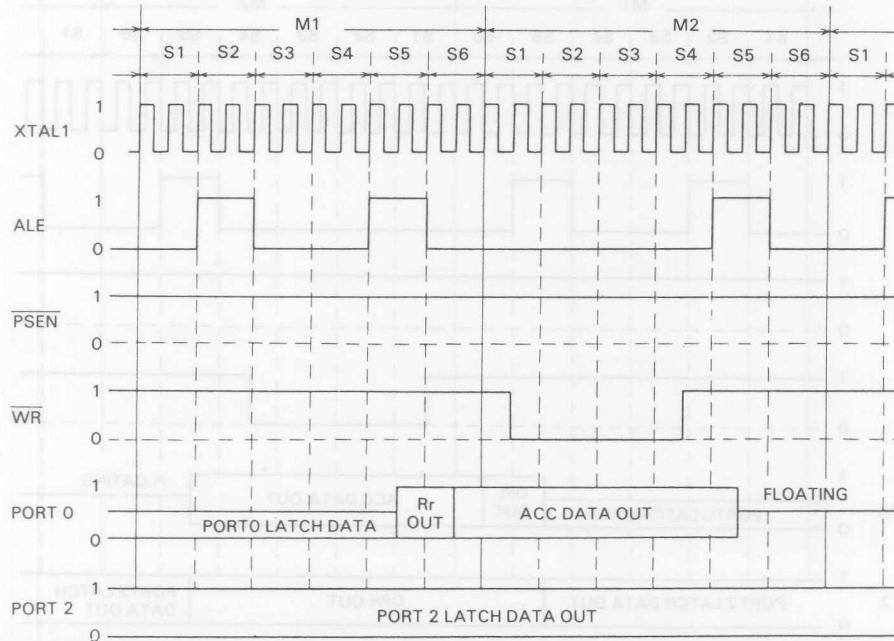


Figure 2-14 MSM80C51 MOVX @Rr, A execution



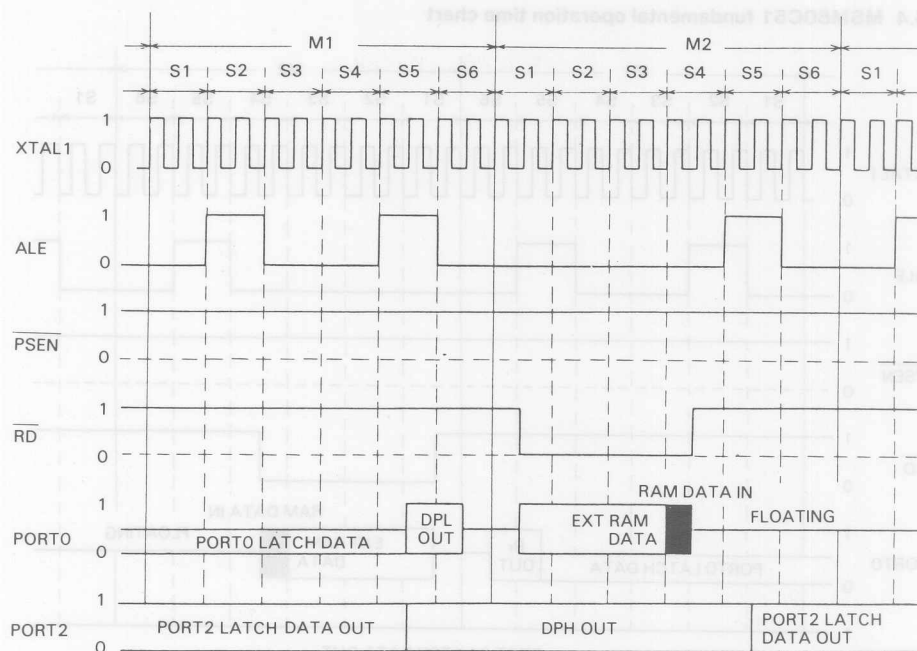


Figure 2-15 MSM80C51 MOVX A, @DPTR execution

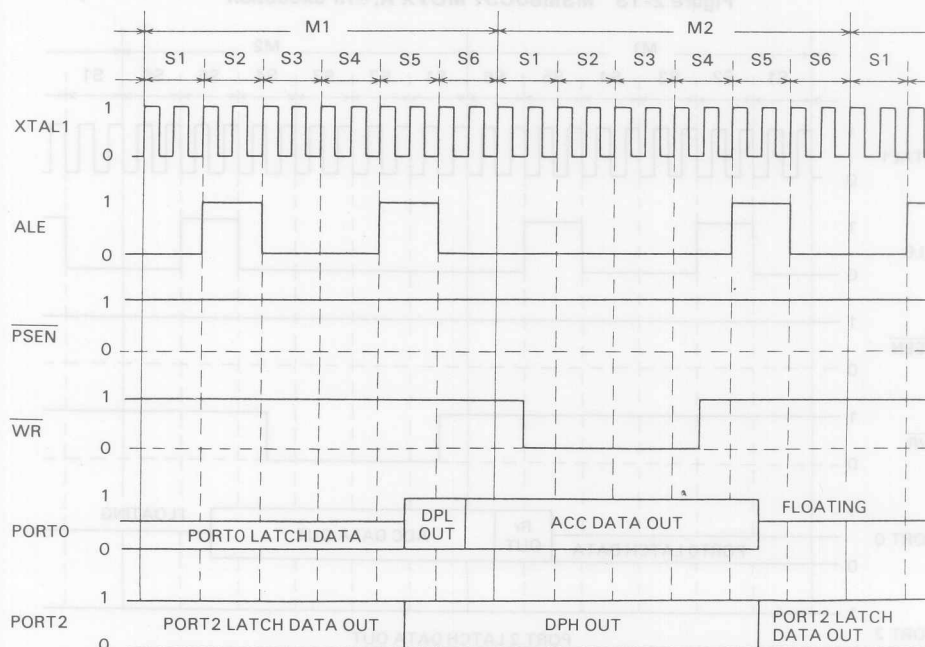


Figure 2-16 MSM80C51 MOVX @DPTR, A execution

## MSM80C31/MSM80C51 SYSTEM CONFIGURATION

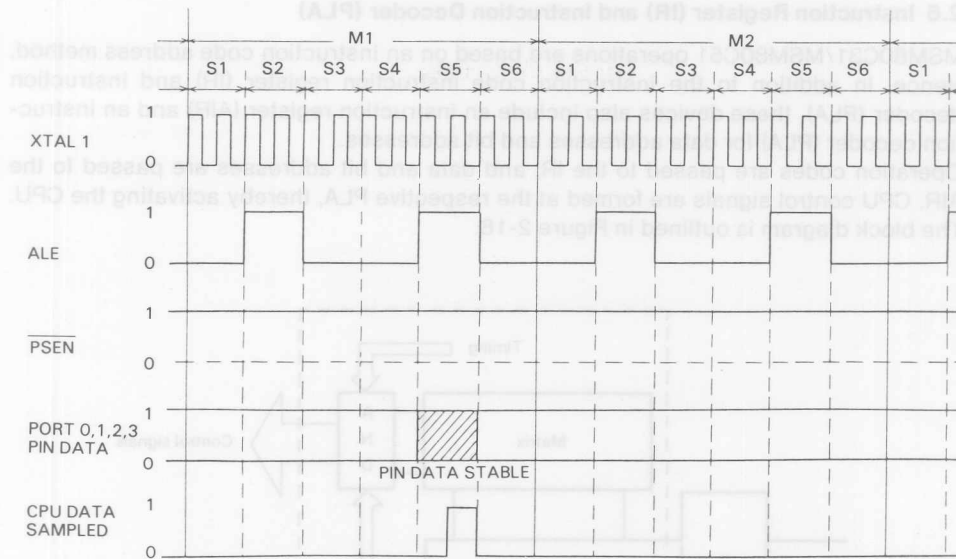


Figure 2-17 MSM80C51 MOV direct, PORT[0, 1, 2, 3] execution

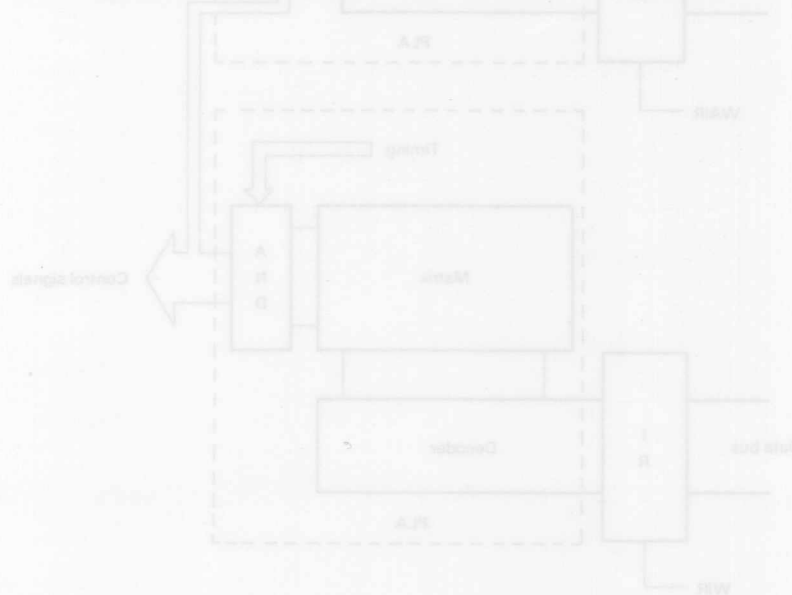


Figure 2-18 IR and PLA block diagram

## 2.6 Instruction Register (IR) and Instruction Decoder (PLA)

MSM80C31/MSM80C51 operations are based on an instruction code address method. Hence, in addition to the instruction code instruction register (IR) and instruction decoder (PLA), these devices also include an instruction register (AIR) and an instruction decoder (PLA) for data addresses and bit addresses.

Operation codes are passed to the IR, and data and bit addresses are passed to the AIR. CPU control signals are formed at the respective PLA, thereby activating the CPU. The block diagram is outlined in Figure 2-18.

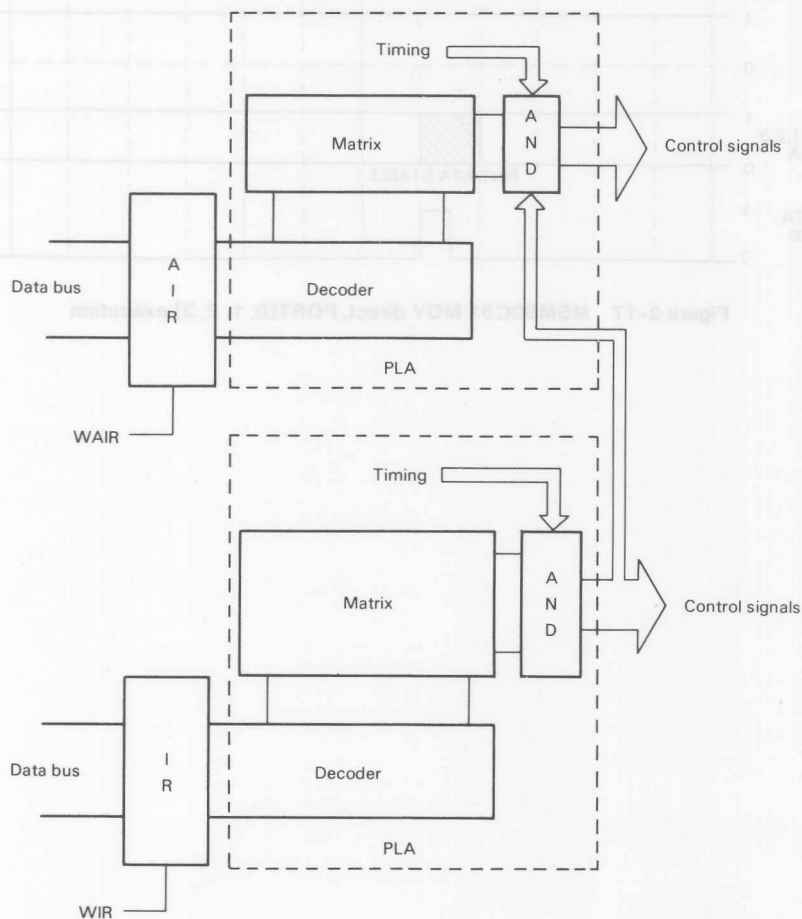


Figure 2-18 IR and PLA block diagram

## 2.7 Arithmetic Operation Section

### (1) Outline

The MSM80C31/MSM80C51 arithmetic operation section consists of

- (1) an arithmetic operation instruction decoder, and
- (2) an arithmetic and logic unit [ALU].

### (2) Arithmetic operation instruction decoder:

Arithmetic operation instructions are passed to the instruction register (IR) and then to the PLA where they are converted into control signals.

The control signals from the PLA are used to control ALU peripheral circuits and ALU internal arithmetic operation (AND, OR, ADD, EOR).

### (3) Arithmetic and logic unit [ALU]:

Upon reception of 8-bit data from one or two data sources the ALU processes that data in accordance with control signals from the PLA. The ALU is capable of executing the following processes:

- Additions and subtractions with and without carry
- Increments (+1) and decrements (-1)
- Bit complements
- Rotations (either direction with and without carry)
- BCD (decimal adjust)
- Carry, auxiliary carry, and overflow signal output
- Multiplications and divisions
- Bit detection
- Exchange of low and high order nibbles
- Logical AND, OR, XOR

If a bit-3 auxiliary carry (AC), a bit-7 carry (CY), or an overflow (OV) is generated as a result of the arithmetic operation executed by the ALU, that result is set in the program status word (PSW).

PSW(D0H)

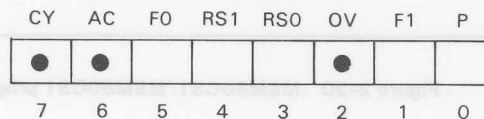


Figure 2-19 Program status word

## 2.8 Program Counter

The MSM80C31/MSM80C51 program counter has a 16-bit configuration PC<sub>0</sub> thru PC<sub>15</sub> as shown in Figure 2-20.

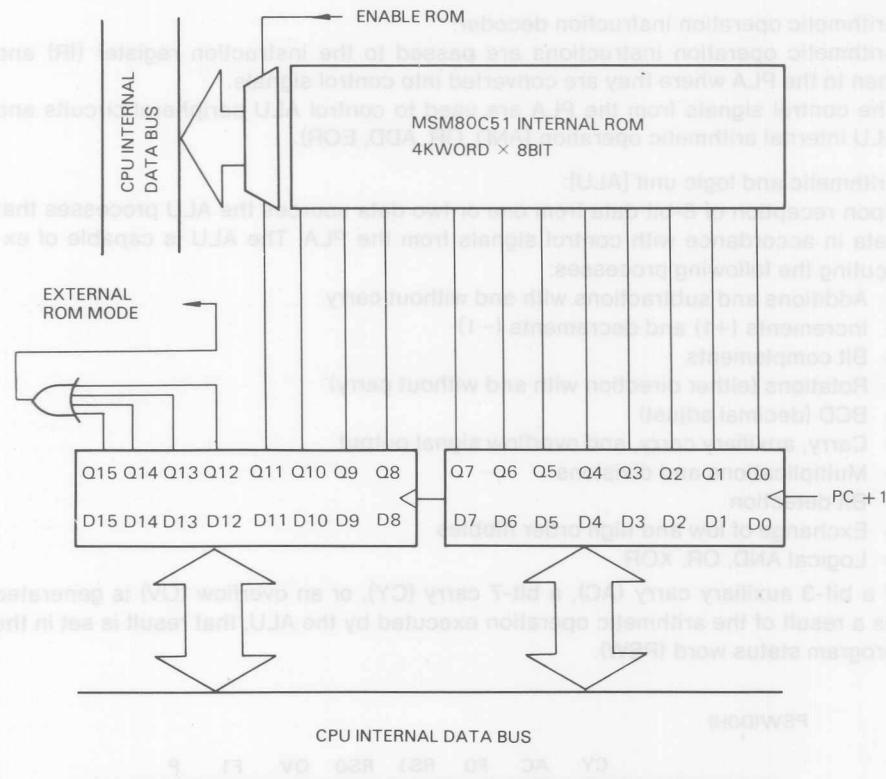


Figure 2-20 MSM80C31/MSM80C51 program counter

This program counter is a binary up-counter which is incremented by 1 each time one byte of instruction code is fetched. When the program counter is counted by 1 after counter contents have reached FFFFH, the counter is returned to 0000H. MSM80C51 is automatically switched to external ROM mode when the counter contents exceed 0FFFH, and executes external instructions.

## 2.9 Program Memory and External Data Memory

### 2.9.1 MSM80C31/MSM80C51 program area and external ROM connections

Since MSM80C31/MSM80C51 are equipped with a 16-bit program counter, these devices can execute programs of up to 64K bytes (including both internal and external programs).

Since the MSM80C31 is not equipped with an internal program ROM, only external instructions are executed. The MSM80C51, on the other hand, is equipped with a 4K byte program ROM which enables it to execute internal instructions from address 0 thru to address 4095. When the address is greater than 4095, external instructions are executed. The program area is outlined in Figure 2-21, and a diagram of ROM connections made when external instructions are executed is shown in Figure 2-22.

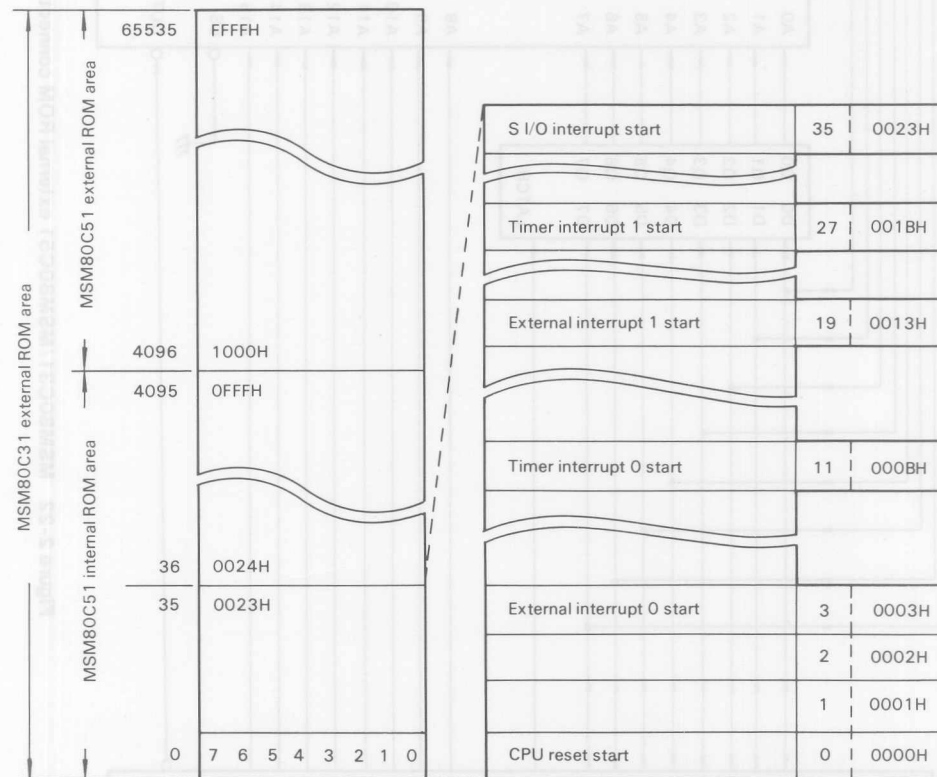


Figure 2-21 MSM80C31/MSM80C51 program area

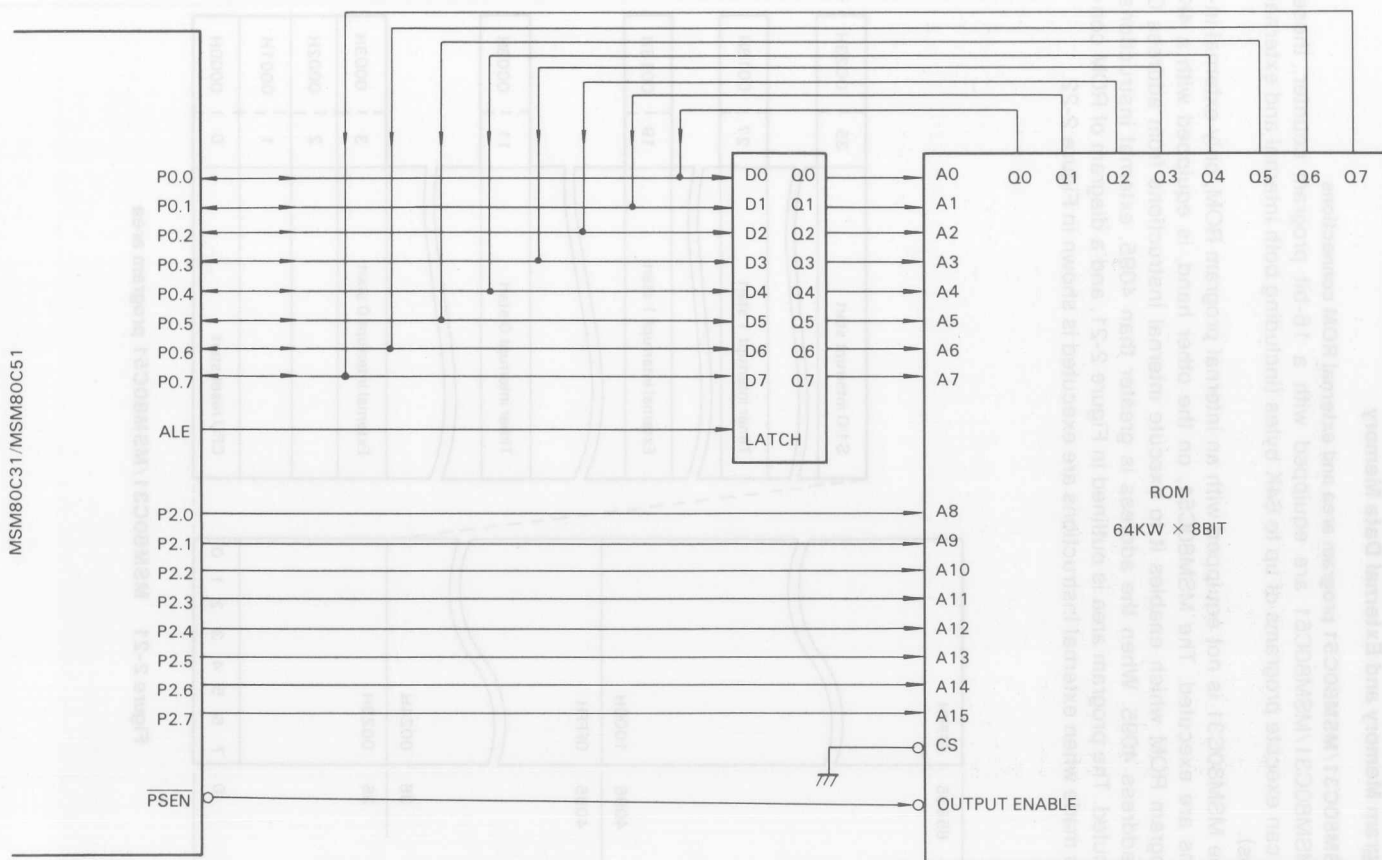


Figure 2-22 MSM80C31/MSM80C51 external ROM connection diagram

### 2.9.2 Procedures and circuit connections used when external data memory (RAM) is read/written by data pointer

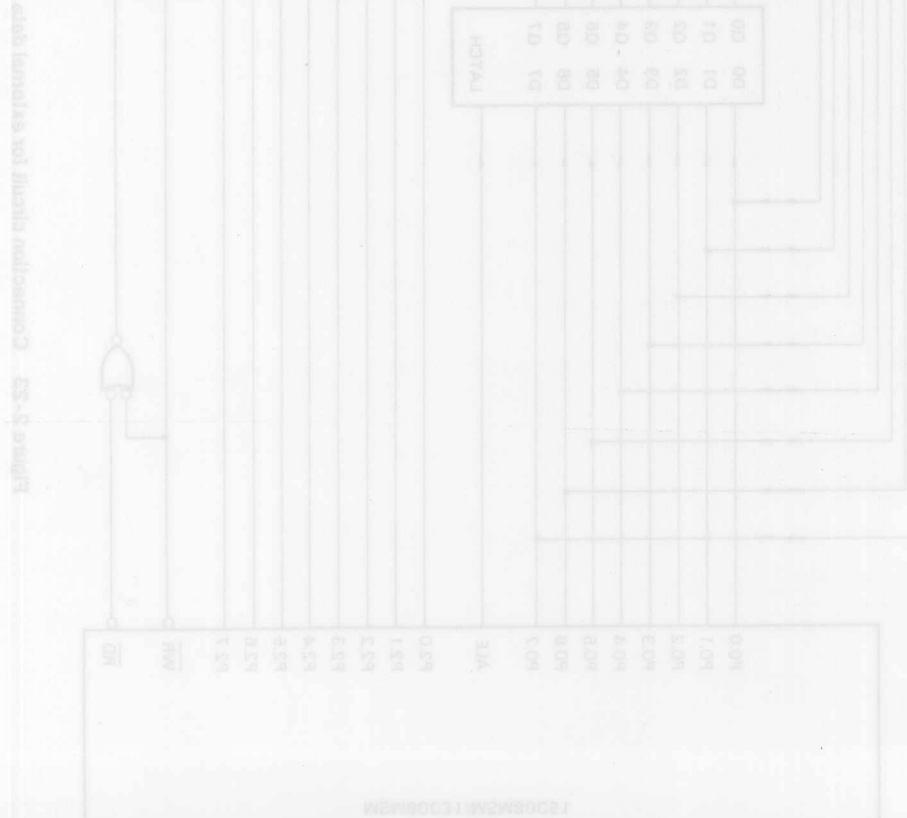
The MSM80C31/MSM80C51 can be connected to an external 64K word $\times$ 8-bit data memory (RAM) when accessing the memory by data pointer (DPTR).

The data pointer (DPTR) consists of DPL and DPH registers. The DPL register contents serves as addresses 0 thru 7 of the external data memory, the DPH register contents serves as addresses 8 thru 15.

The MOVX @DPTR, A instruction is used when accumulator contents are transferred to an external data memory, and the MOVX A, @DPTR instruction is used when external data memory contents are transferred to the accumulator. The external data memory connection diagram is shown in Figure 2-23 and the external data memory access time chart is shown in Figure 2-24.

When the data pointer indirect external memory instruction is executed, the CPU passes the DPL register contents to port 0, and the port 0 contents are then latched externally by the ALE signal. Data stored in the latch serves as the lower order addresses 0 thru 7 of the external data memory (RAM), while the DPH register contents output from port 2 serves as the higher order addresses 8 thru 15 for direct addressing of the external data memory.

The WR or RD external data memory control signal is subsequently generated by the CPU to enable transfer of data between port 0 and the external data memory.





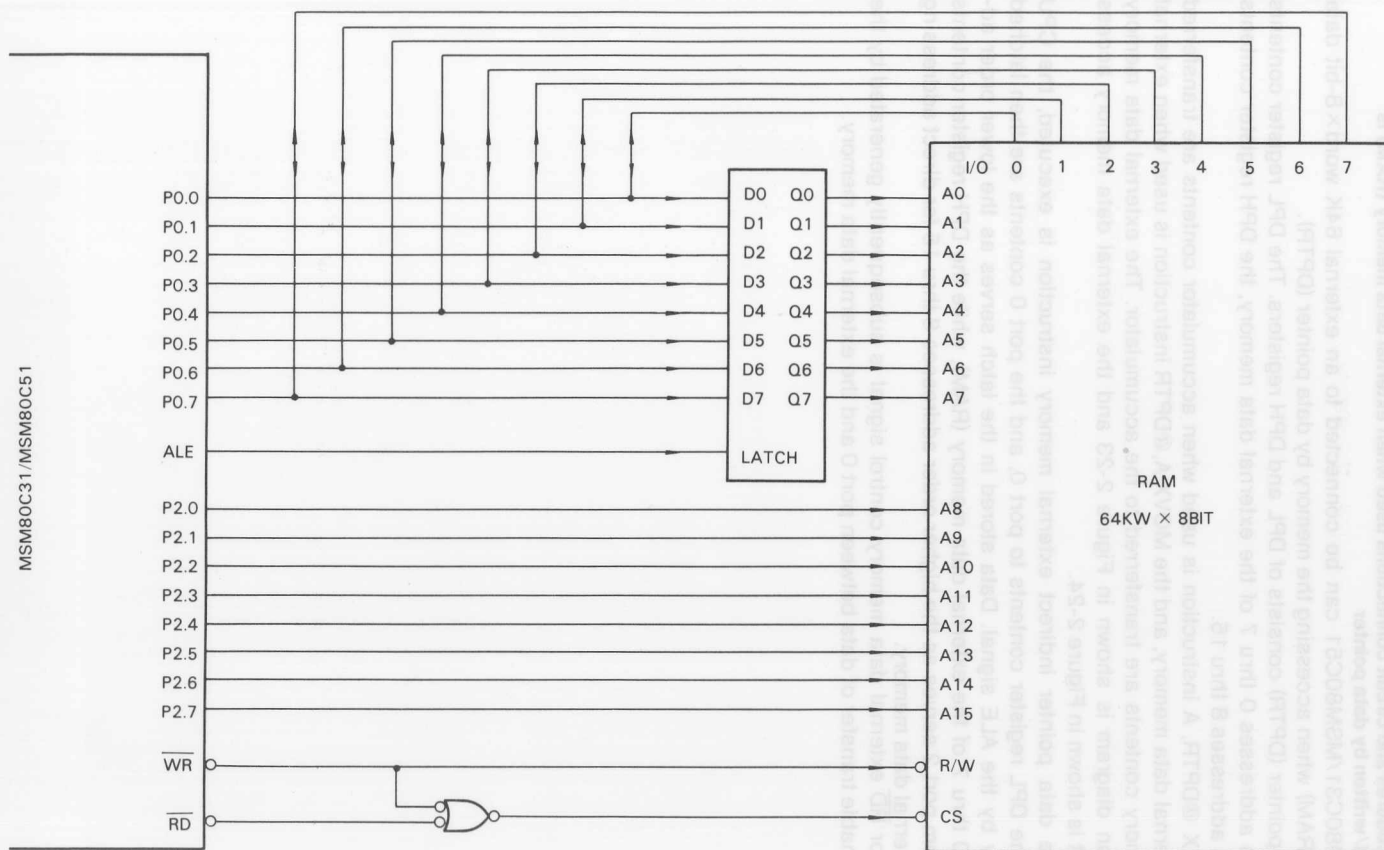


Figure 2-23 Connection circuit for external data memory addressed by DPTR

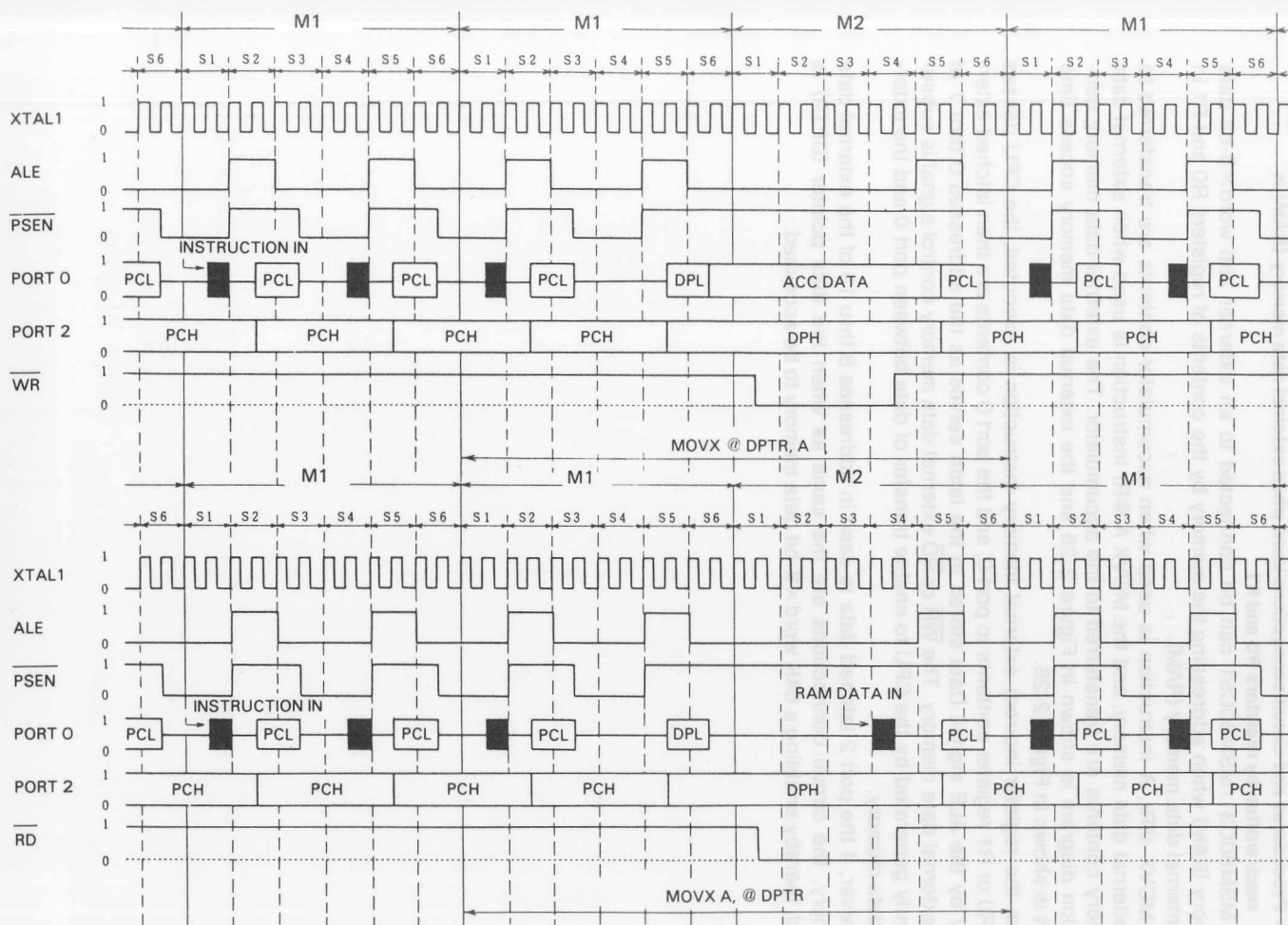


Figure 2-24 DPTR external data memory R/W timing

### 2.9.3 Procedures and circuit connections used when external data memory (RAM) is read/written by registers R0 and R1

The MSM80C31/MSM80C51 can be connected to an external 256 word $\times$ 8-bit data memory (RAM) when addressing the memory by the contents of registers R0 and R1 in the internal data memory (RAM).

The MOVX @Rr, A instruction is used when accumulator contents are transferred to an external data memory, and the MOVX A, @Rr instruction is used when external data memory contents are transferred to the accumulator. The external data memory connection diagram is shown in Figure 2-25 and the external data memory access time chart is shown in Figure 2-26.

When the register indirect external memory instruction is executed, the CPU passes the R0 or R1 register contents to port 0, and the port 0 contents are then latched externally by the ALE signal. Data stored in the latch serves as the addresses 0 thru 7 of the external data memory. The  $\overline{WR}$  or  $\overline{RD}$  external data memory control signal is subsequently generated by the CPU to enable transfer of data between port 0 and the external data memory.

However, if the port 2 latched data is used in addresses 8 thru 15 of the external data memory, the circuit connections are the same as when the data pointer (DPTR) is used, thereby enabling a 64K word $\times$ 8-bit data memory to be accessed.



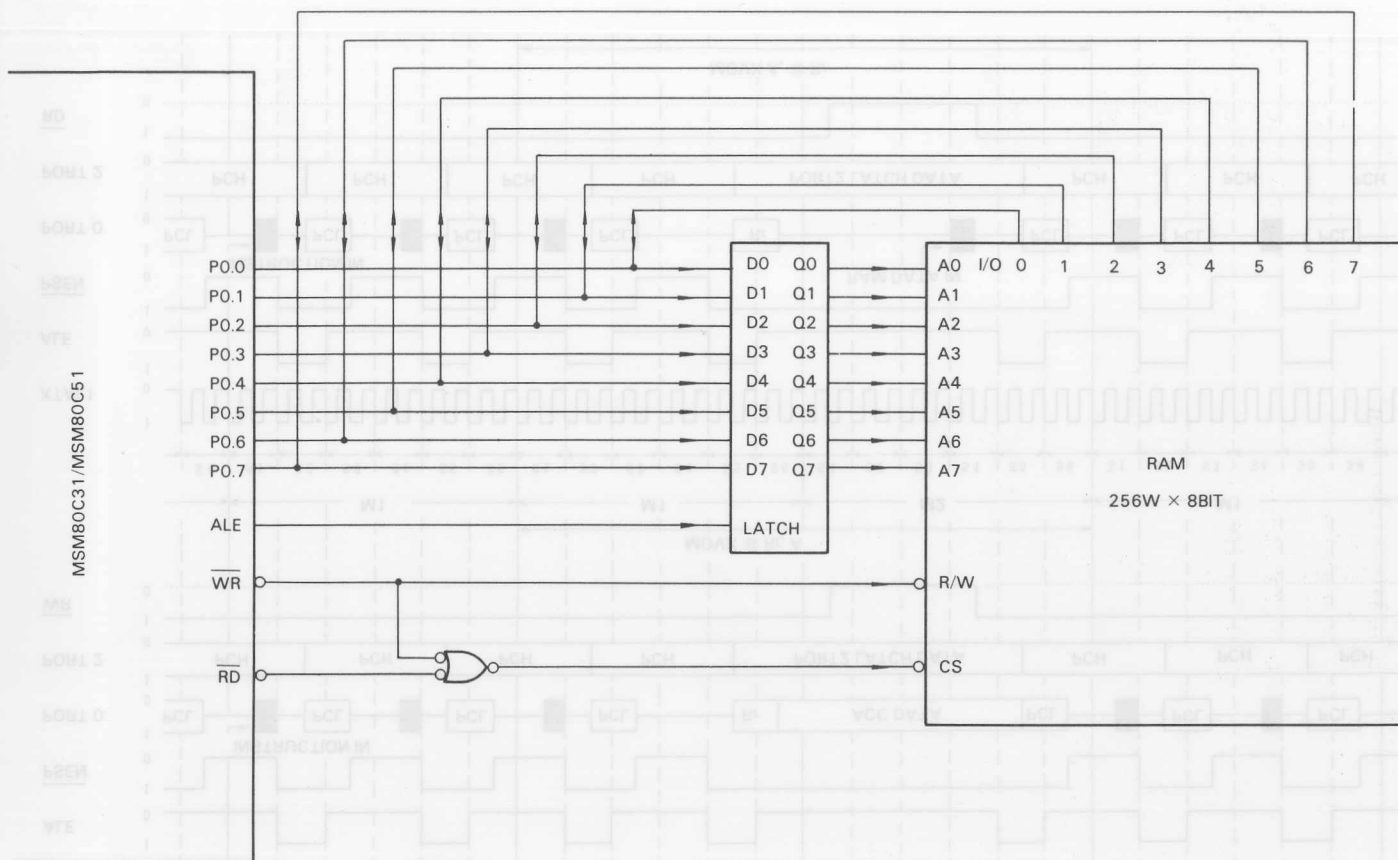


Figure 2-25 Connection circuit for external data memory addressed by register R0 or R1

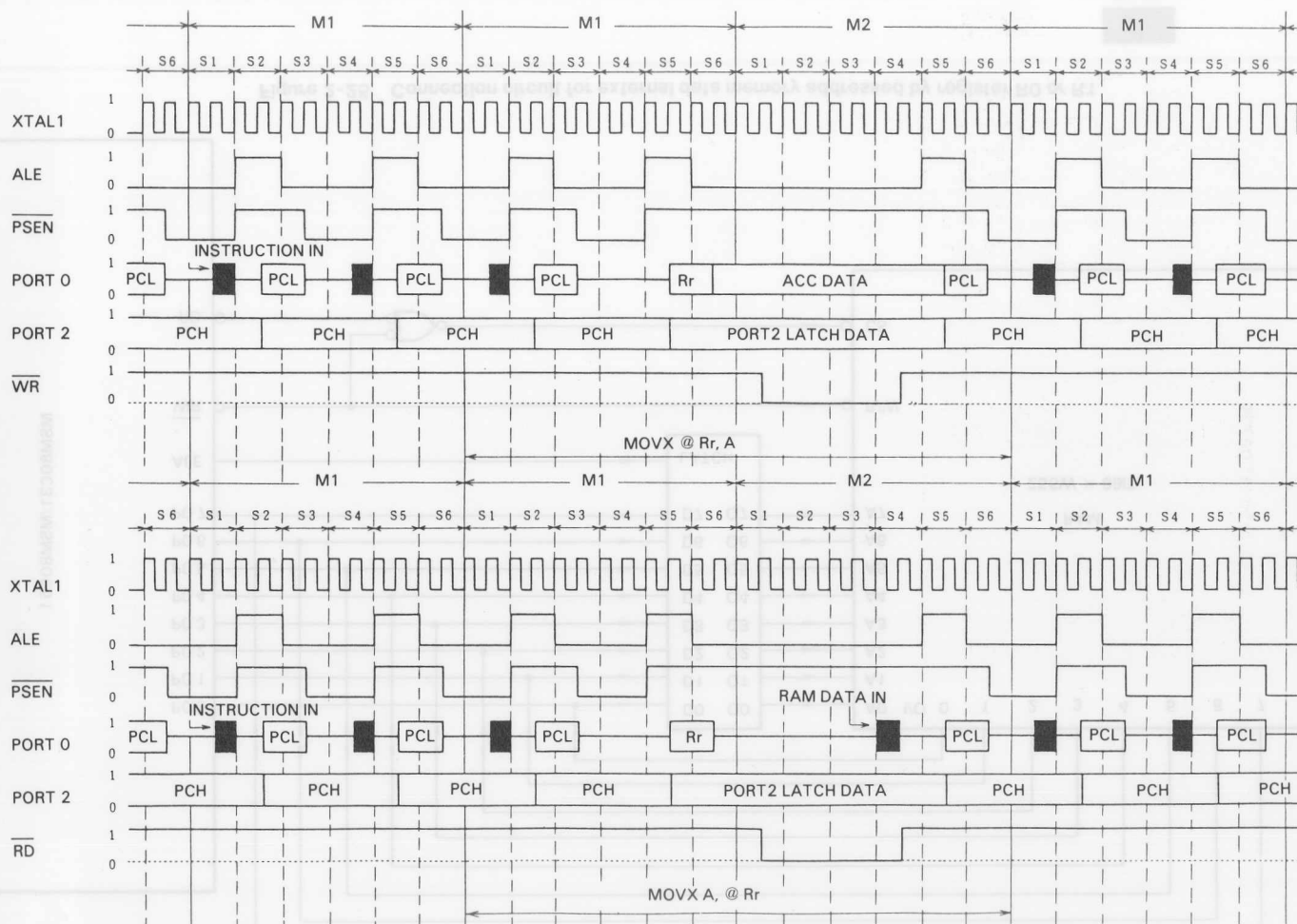


Figure 2-26 Register R0/R1 external data memory R/W timing

# 3. MSM80C31/ MSM80C51 CONTROL

3



Figure 3-1 Crystal resonator connection diagram

\* The XTAL1, 2 frequency depends on Vcc

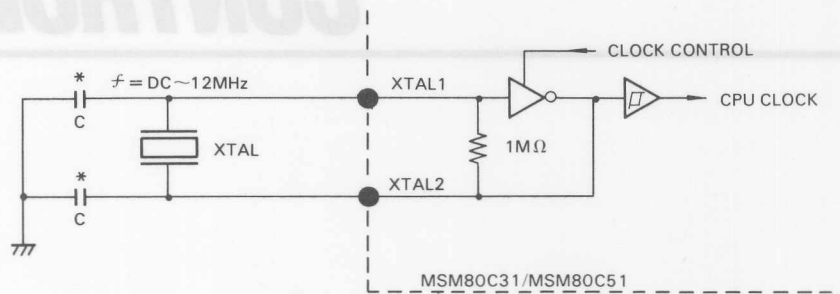
\* The capacity of the compensating capacitor depends on the crystal resonator

### 3. MSM80C31/MSM80C51 CONTROL

#### 3.1 Oscillators: XTAL 1 XTAL 2

An oscillator is formed by connecting a crystal or ceramic resonator between the XTAL1 and XTAL2 pins of the MSM80C31/MSM80C51 devices.

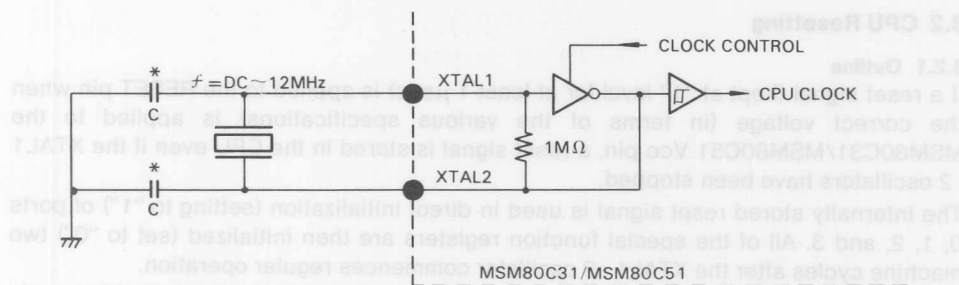
If an external clock is applied to XTAL1 · 2, the input should be at 50% duty and C-MOS level.



\* The capacity of the compensating capacitor depends on the crystal resonator.

\* The XTAL1 · 2 frequency depends on Vcc.

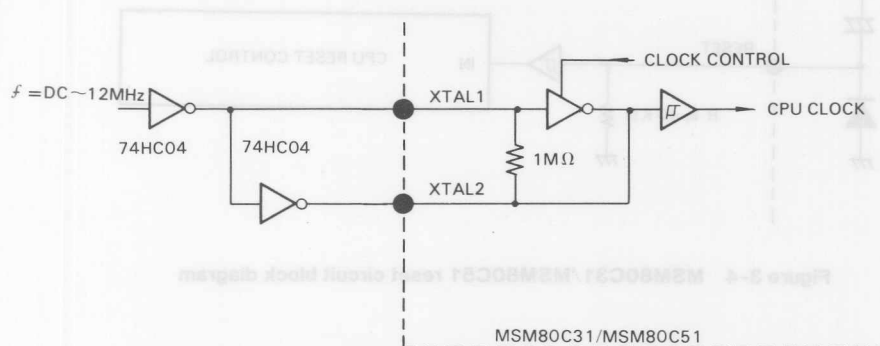
Figure 3-1 Crystal resonator connection diagram



\* The capacity of the compensating capacitor depends on the ceramic resonator.

\* The XTAL1 · 2 frequency depends on Vcc.

**Figure 3-2 Ceramic resonator connection diagram**



\* The external clock frequency depends on Vcc.

\* Supply of 50% duty clock

**Figure 3-3 External clock supply circuit**



### 3.2 CPU Resetting

#### 3.2.1 Outline

If a reset signal (kept at "1" level for at least 1  $\mu\text{sec}$ ) is applied to the RESET pin when the correct voltage (in terms of the various specifications) is applied to the MSM80C31/MSM80C51 Vcc pin, a reset signal is stored in the CPU even if the XTAL1  $\cdot$  2 oscillators have been stopped.

The internally stored reset signal is used in direct initialization (setting to "1") of ports 0, 1, 2, and 3. All of the special function registers are then initialized (set to "0") two machine cycles after the XTAL1  $\cdot$  2 oscillator commences regular operation.

When the reset is released, instruction execution is started in the third machine cycle if the reset signal is changed from "1" level to "0" level before the M1  $\cdot$  S1 signal leading edge, and in the fifth machine cycle if the reset signal is changed from "1" to "0" after the leading edge.

The reset circuit block diagram is shown in Figure 3-4, the reset start time charts in Figures 3-5 and 3-6, and the reset release time charts in Figures 3-7 and 3-8.

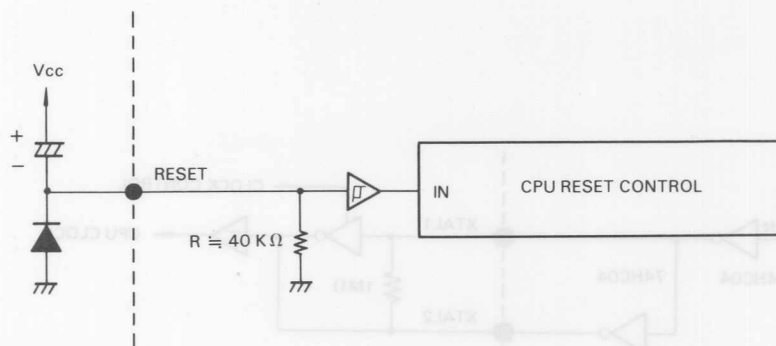


Figure 3-4 MSM80C31/MSM80C51 reset circuit block diagram

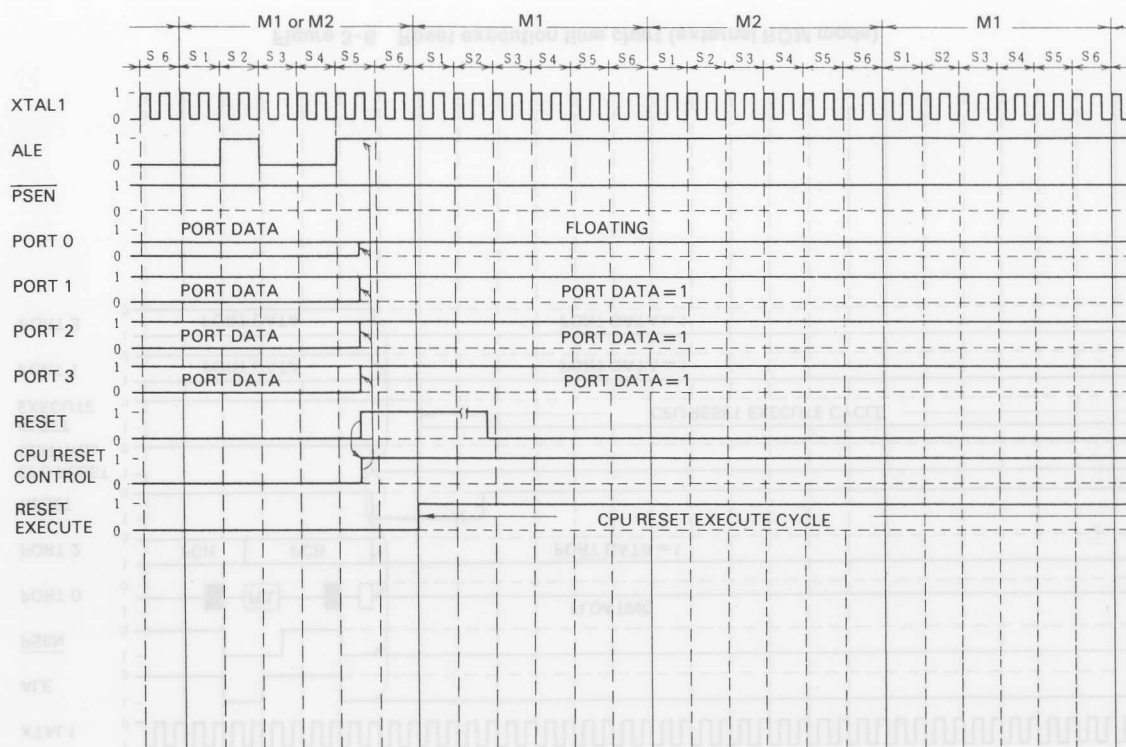


Figure 3-5 Reset execution time chart (internal ROM mode)

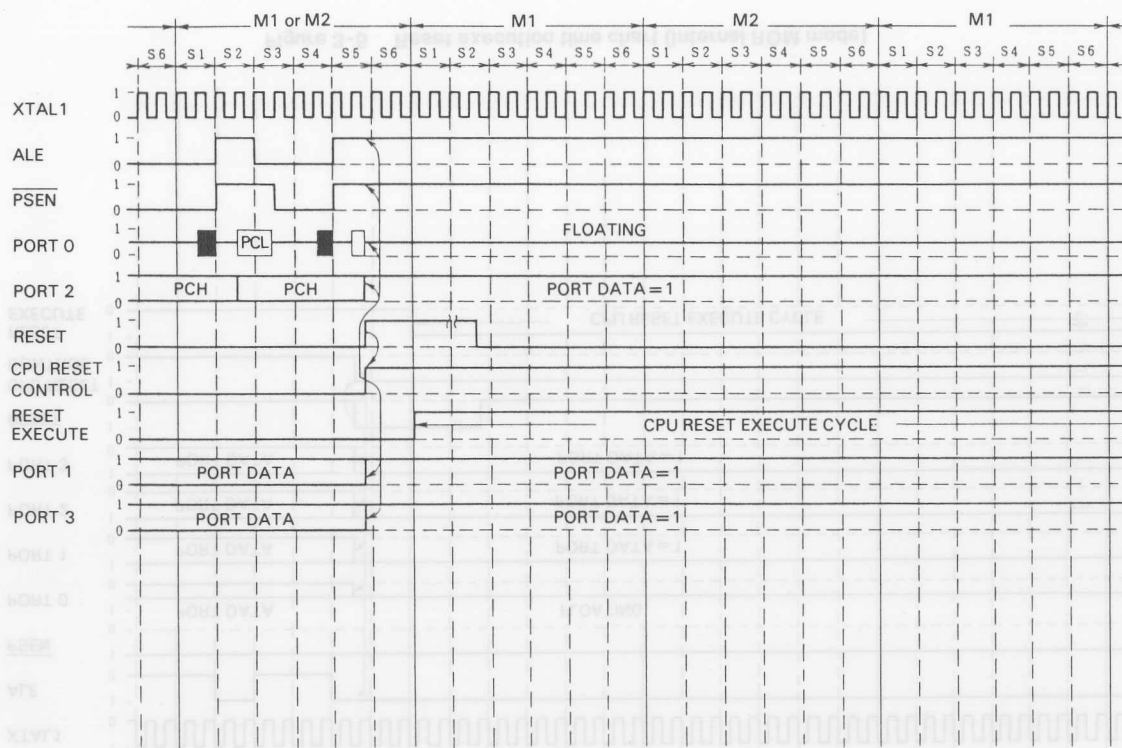


Figure 3-6 Reset execution time chart (external ROM mode)

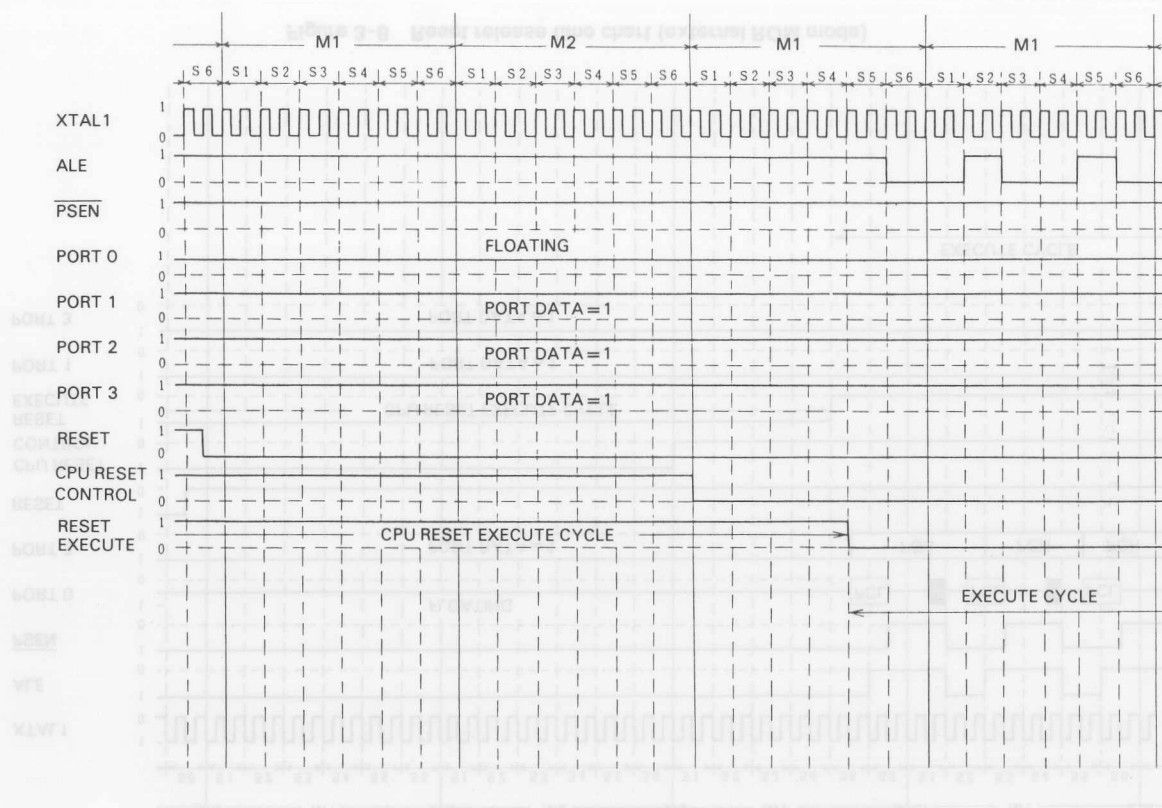


Figure 3-7 Reset release time chart (internal ROM mode)

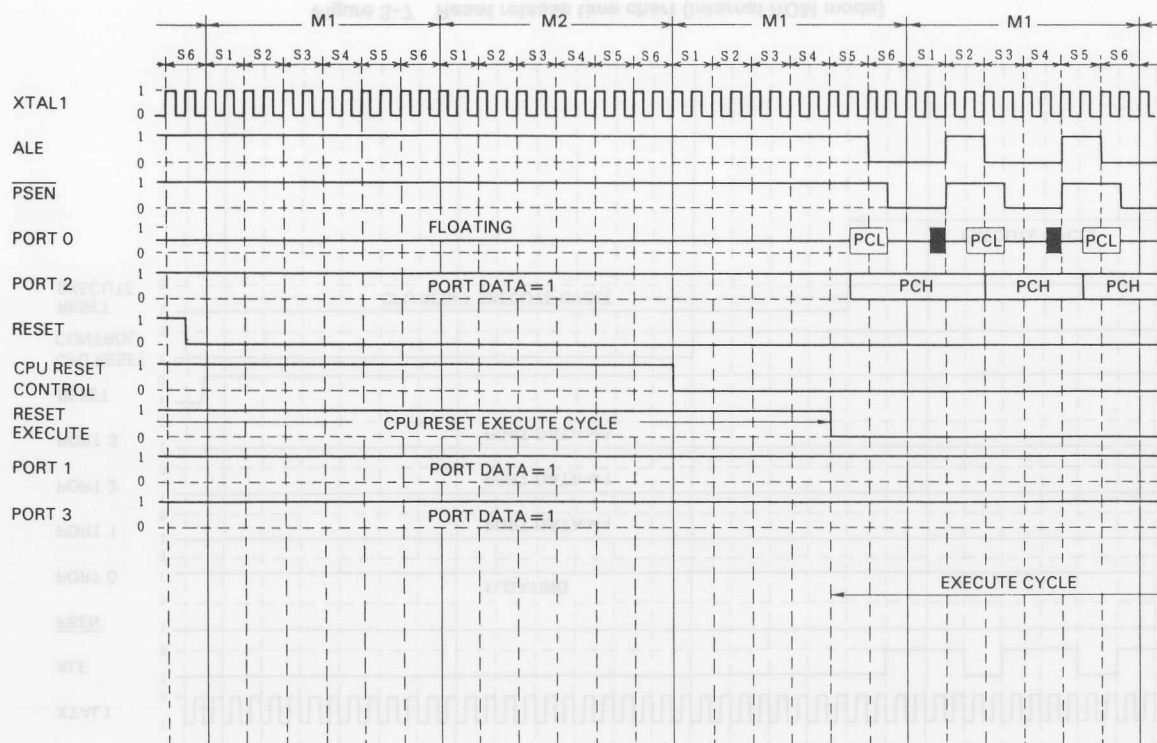


Figure 3-8 Reset release time chart (external ROM mode)

### 3.2.2 Reset schmitt trigger circuit

The Schmitt trigger circuit connected to the RESET pin shown in Figure 3-4 operates in the following way when the Vcc power supply voltage is +5 V.

If the voltage of the input reset signal applied to the RESET pin exceeds 3 V when the level of that signal is changed from "0" to "1", the Schmitt trigger output level is changed from "0" to "1", and the reset signal is set in the CPU reset control circuit, resulting in the reset operation being started by the CPU.

The CPU reset state is released when the "1" level on the RESET pin is changed to "0". An input signal level below 1.5 V is regarded as "0" level, and the Schmitt trigger output level is changed from "1" to "0". When the reset signal is changed to "0" level, the CPU reset control circuit is ready for reset release. The Schmitt trigger circuit operation time chart for changes in the reset input voltage is outlined in Figure 3-9.

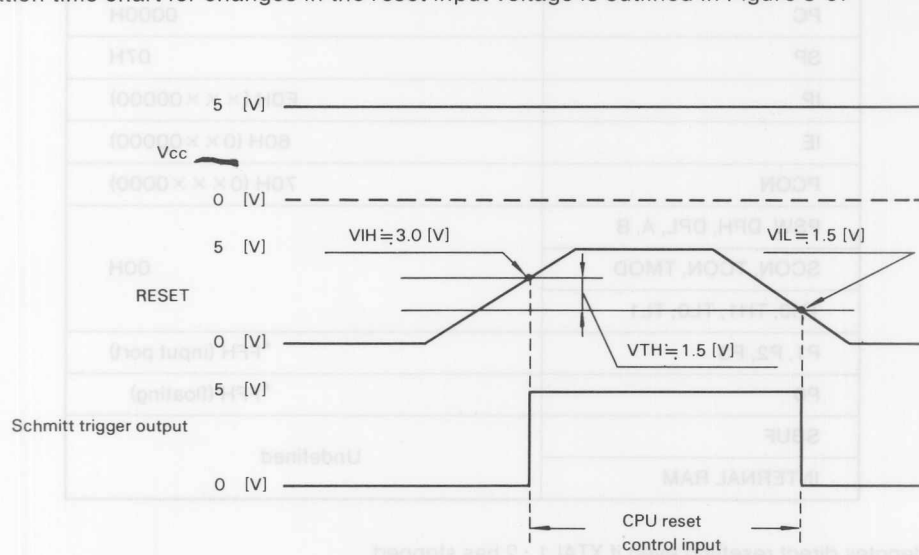


Figure 3-9 Reset schmitt trigger gate detector time chart

### 3.2.3 CPU reset internal status

When a reset signal is applied to the CPU with normal voltage applied to the MSM80C31/MSM80C51 Vcc power supply pin, the I/O port is set to "1" (input mode) directly even if XTAL1 · 2 has stopped. The CPU is then reset after normal XTAL1 · 2 oscillation has been resumed. The internal CPU status when the CPU is reset is shown in Table 3-1.

**Table 3-1 MSM80C31/MSM80C51 reset internal status**

Register name	Register reset status
PC	0000H
SP	07H
IP	E0H (× × × 00000)
IE	60H (0 × × 00000)
PCON	70H (0 × × × 0000)
PSW, DPH, DPL, A, B	00H
SCON, TCON, TMOD	
TH0, TH1, TL0, TL1	
P1, P2, P3	*FFH (input port)
P0	*FFH (floating)
SBUF	Undefined
INTERNAL RAM	

\* denotes direct resetting even if XTAL1 · 2 has stopped.

### 3.3 $\overline{EA}$ (CPU Memory Separate)

#### 3.3.1 Outline

The function of the  $\overline{EA}$  pin is to determine whether a CPU internal program memory (ROM) instruction is to be executed, or an external program instruction is to be executed.

(1) Internal ROM mode

If the  $\overline{EA}$  pin is connected to Vcc and a "1" reset signal is applied to the RESET pin to reset the CPU, an internal program memory (ROM) is executed from address 0. (MSM80C51 only)

(2) External ROM mode

If the  $\overline{EA}$  pin is connected to Vss and a "1" reset signal is applied to the RESET pin to reset the CPU, an external program memory is executed from address 0.





## 4. MSM80C31/MSM80C51 INTERNAL SPECIFICATIONS

Function Registers (SFR)

# 4. MSM80C31/ MSM80C51 INTERNAL SPECIFICATIONS

4

(RAM) and special function register address space layout is shown in Figure 4-1.

P0, TCON, SC0N, IE, IP, PSW, ACC, and B, a total of eleven registers. The data memory

bits being specified by the three lower order bits and the data memory (RAM) or special

function register (ACC, B, TCON, P0, ... ) by the five higher order bits.

Special function registers which can be specified by bit designation.

The bit addresses which can be specified in data memory (RAM) are addresses 20

bits being specified by the three lower order bits and the data memory (RAM) or special

function register (ACC, B, TCON, P0, ... ) by the five higher order bits.

Special function registers are located between addresses 80 thru FFH, and can also

be specified directly by data address. Bit addresses consist of eight bits, the operation

bits being specified by the three lower order bits and the data memory (RAM) or special

function register (ACC, B, TCON, P0, ... ) by the five higher order bits.

The data memory contains 128 bytes in data addresses 00 thru 7FH, and can be spec-

ified directly by data address.

addresses consist of eight bits, and range from 00 to FFH in binary (which correspond to

0 thru 255 in decimal). All data memory (RAM) and special function registers (ACC, B,

TCON, P0, ... ) exist in these 256 locations.

## 4. MSM80C31/MSM80C51 INTERNAL SPECIFICATIONS

### 4.1 Internal Data Memory (RAM) and Special Function Registers (SFR)

#### 4.1.1 Outline

MSM80C31/MSM80C51 operation is based on an instruction code address method where operations are specified in an instruction code (OP) section, and the data memory (RAM) and special function registers (ACC, B, TCON, P0, ..... ) are specified directly by part of the instruction code and the second or third byte of data following that instruction code.

According to this instruction code address method, all eight bits of data in the data memory and the special function register may be specified, or one bit of data memory and one bit of data in the special function register may be specified. Direct designation of all eight bits of data is called data addressing, and direct designation of one bit of data is called bit addressing.

Since these CPU devices specify data memory (RAM) and special function register contents by the above method, specific addresses are assigned to the respective CPU data memory (RAM) and special function registers (ACC, B, TCON, P0, .... ). Data addresses consist of eight bits, and range from 00 to FFH in binary (which correspond to 0 thru 255 in decimal). All data memory (RAM) and special function registers (ACC, B, TCON, P0, .... ) exist in these 256 locations.

The data memory contains 128 bytes in data addresses 00 thru 7FH, and can be specified directly by data address.

Special function registers are located between addresses 80 thru FFH, and can also be specified directly by data address. Bit addresses consist of eight bits, the operation bits being specified by the three lower order bits and the data memory (RAM) or special function register (ACC, B, TCON, P0, .... ) by the five higher order bits.

The bit addresses which can be specified in data memory (RAM) are addresses 20 thru 2FH. Other areas cannot be specified by bit designation.

Special function registers which can be specified by bit address include P0, P1, P2, P3, TCON, SCON, IE, IP, PSW, ACC, and B, a total of eleven registers. The data memory (RAM) and special function register address space layout is shown in Figure 4-1.

# MSM80C31/MSM80C51 INTERNAL SPECIFICATIONS

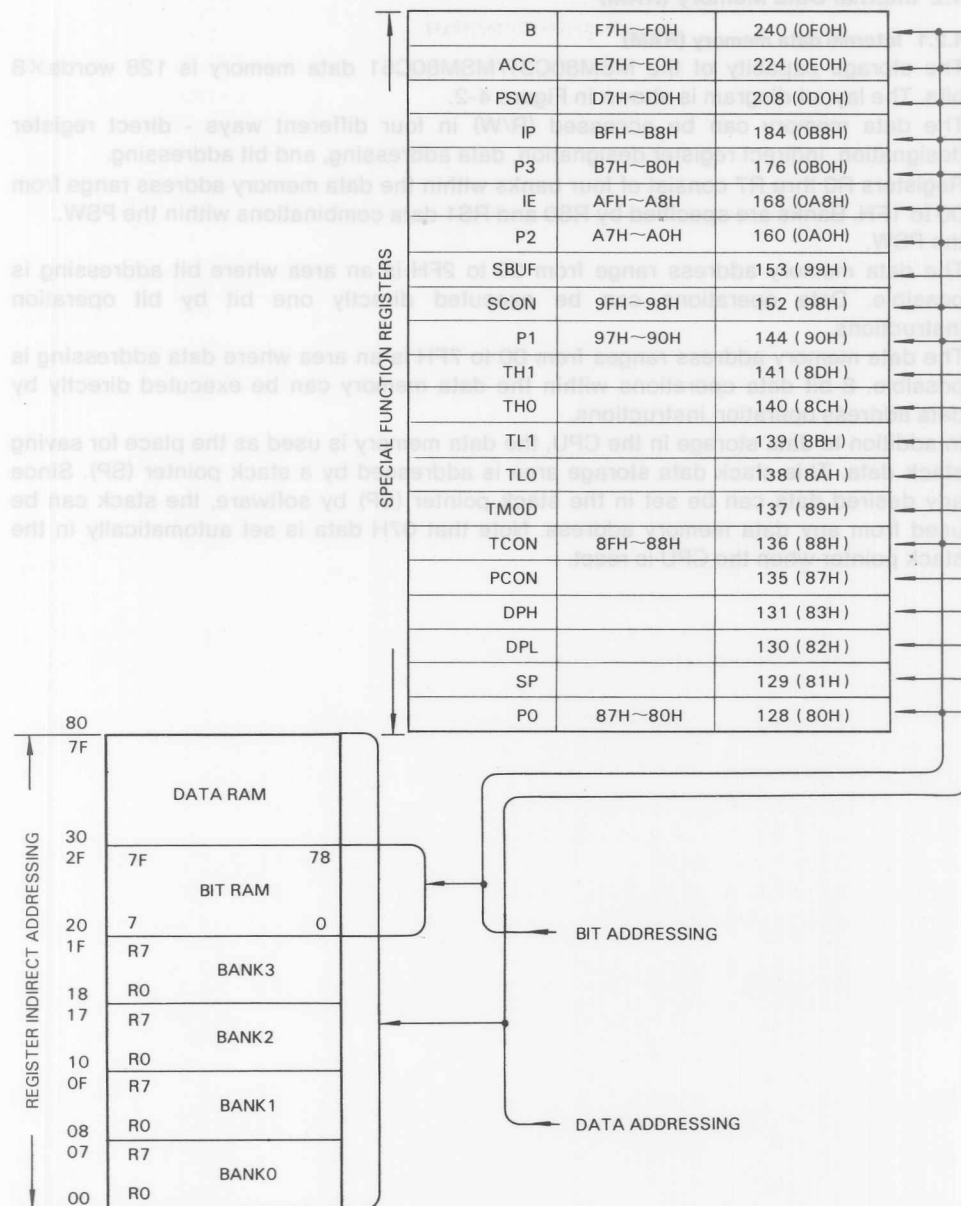


Figure 4-1 Data memory and special function register layout

## 4.2 Internal Data Memory (RAM)

### 4.2.1 Internal data memory (RAM)

The storage capacity of the MSM80C31/MSM80C51 data memory is 128 words $\times$ 8 bits. The layout diagram is shown in Figure 4-2.

The data memory can be accessed (R/W) in four different ways - direct register designation, indirect register designation, data addressing, and bit addressing.

Registers R0 thru R7 consist of four banks within the data memory address range from 00 to 1FH. Banks are specified by RS0 and RS1 data combinations within the PSW. the PSW.

The data memory address range from 20 to 2FH is an area where bit addressing is possible. Data operations can be executed directly one bit by bit operation instructions.

The data memory address ranges from 00 to 7FH is an area where data addressing is possible. 8-bit data operations within the data memory can be executed directly by data address operation instructions.

In addition to data storage in the CPU, the data memory is used as the place for saving stack data. This stack data storage area is addressed by a stack pointer (SP). Since any desired data can be set in the stack pointer (SP) by software, the stack can be used from any data memory address. Note that 07H data is set automatically in the stack pointer when the CPU is reset.



Figure 4-1 Data memory and special function register layout

# MSM80C31/MSM80C51 INTERNAL SPECIFICATIONS

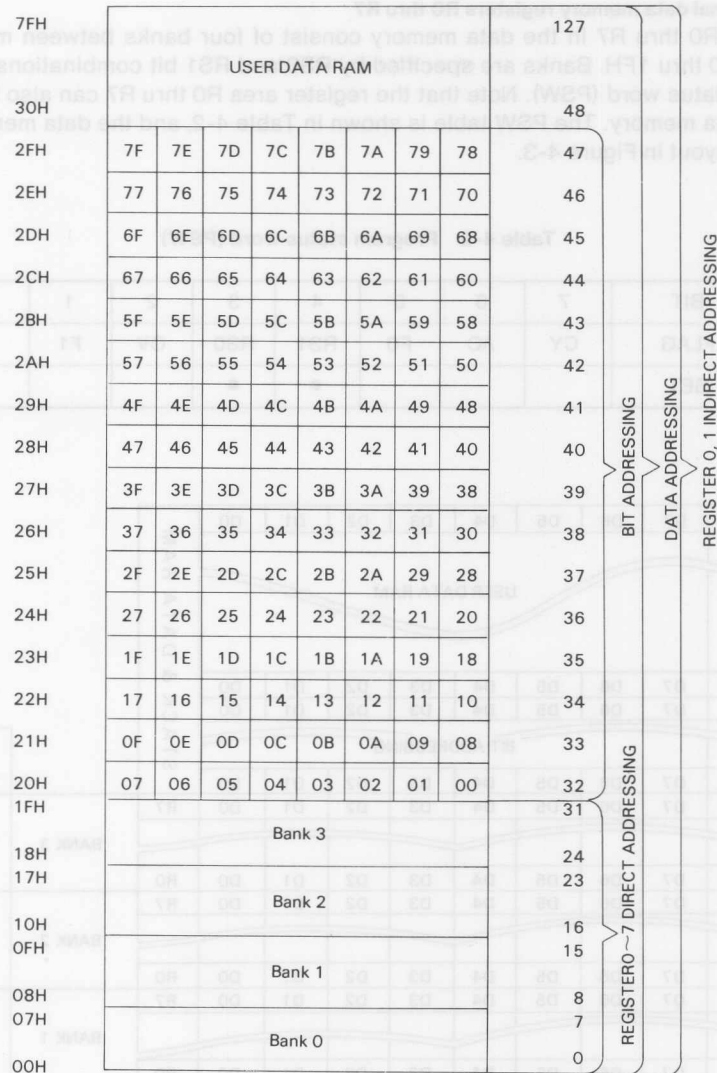


Figure 4-2 RAM layout diagram

#### 4.2.2 Internal data memory registers R0 thru R7

Registers R0 thru R7 in the data memory consist of four banks between memory addresses 00 thru 1FH. Banks are specified by RS0 and RS1 bit combinations within the program status word (PSW). Note that the register area R0 thru R7 can also be used as normal data memory. The PSW table is shown in Table 4-2, and the data memory register bank layout in Figure 4-3.

Table 4-2 Program status word (PSW)

BIT	7	6	5	4	3	2	1	0
FLAG	CY	AC	F0	RS1	RS0	OV	F1	P
SET				•	•			

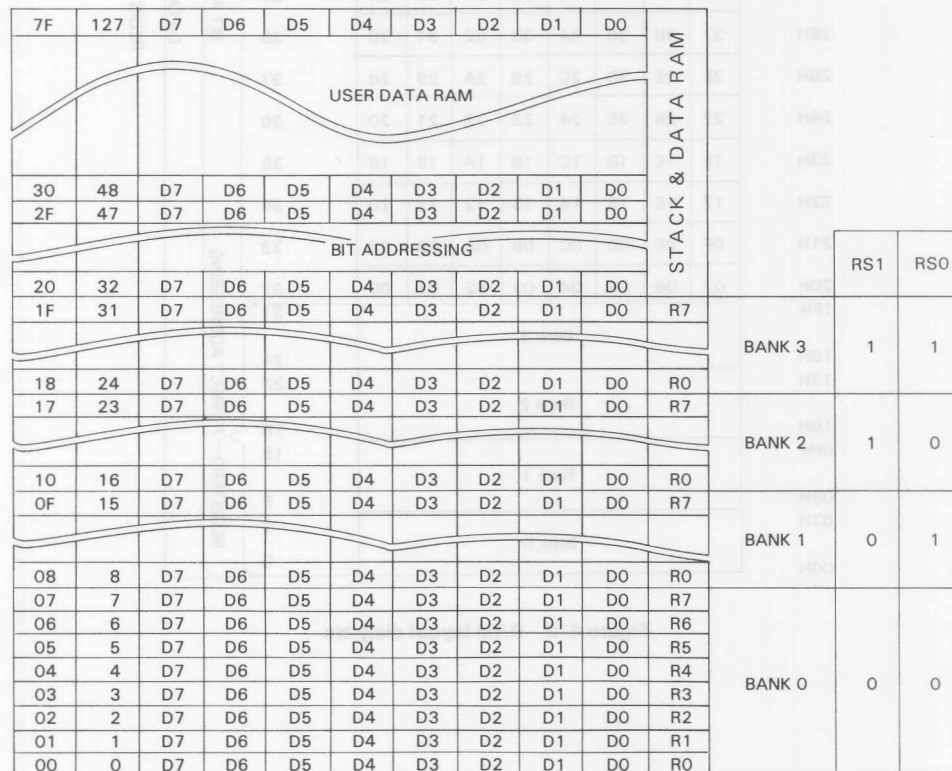


Figure 4-3 Internal data memory register bank layout

#### 4.2.3 Stack

The stack data save area is in the internal data memory (RAM), and is specified by stack pointer (SP 81H).

Although 07H data is automatically set in the stack pointer when the CPU is reset, any desired data can be set by software to enable the data memory from any address to be used as the stack.

Two addresses of data memory are used when the stack is used by interrupt or CALL instruction, and a single address is used when the PUSH instruction is used.

The status where an interrupt is generated and the program counter contents are saved in the stack when the stack pointer contents are 5FH, and accumulator contents are pushed during interrupt routine are shown in Table 4-3. The stack status up to completion of interrupt processing upon execution of POP and RETI instructions is also included.

Table 4-3 Stack storage data RAM layout

Stack processing	Stack pointer	RAM data bit							
		7	6	5	4	3	2	1	0
Before execution	5FH	D7	D6	D5	D4	D3	D2	D1	D0
Interrupt process	60H	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
	61H	PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8
PUSH process (ACC)	62H	A7	A6	A5	A4	A3	A2	A1	A0
POP process (ACC)	62H	A7	A6	A5	A4	A3	A2	A1	A0
RETI process	61H	PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8
	60H	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
After execution	5FH	D7	D6	D5	D4	D3	D2	D1	D0



### 4.3 Internal Data Memory (RAM) Operating Procedures

#### 4.3.1 Internal data memory indirect addressing

Operation of the internal data memory indirect increment instruction is described here as an example. This instruction (INC @Rr) is a 1-byte 1-machine cycle instruction (see Figure 4-4), and the indirect address register is specified by instruction code bit 0 data r where r denotes either register 0 or 1 in the register group specified by PSW RS0 and RS1 bank data. Register 0 is specified when the r data is 0, and register 1 is specified when the data is 1.

When this instruction is executed, register data is read from the specified register 0 or 1, and the read out register data is written into the data pointer of the data memory.

The data memory contents specified by the data pointer are read by the CPU into a temporary register. Then a subsequent increment (+1) by the ALU is followed by a return to the data memory at the address specified by the data pointer. In this way, the contents of the data memory at the address specified by the contents of R0 or R1 are incremented.

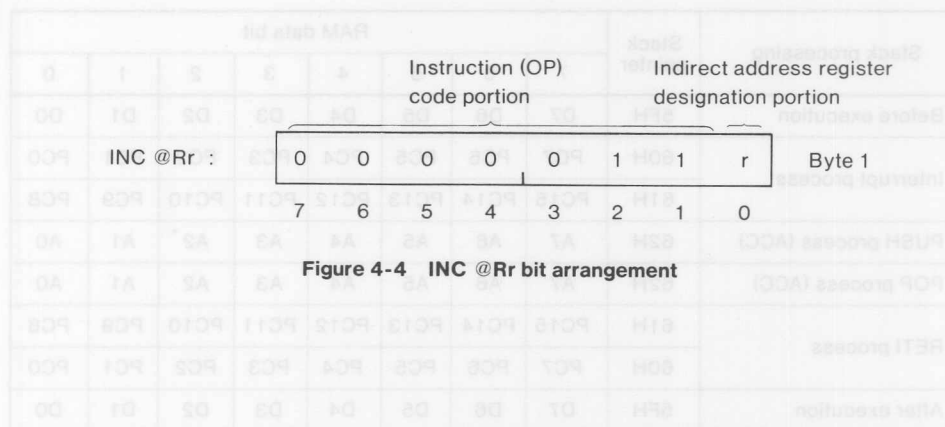


Figure 4-4 INC @Rr bit arrangement

#### 4.3.2 Internal data memory register R0 thru R7 designation

Operation of the internal data memory register direct decrement instruction is described here as an example. This instruction (DEC Rr) is a 1-byte 1-machine cycle instruction (see Figure 4-5). Register R0 thru R7 is specified by  $r_0$ ,  $r_1$ , and  $r_2$  data of instruction code bit 0, 1, and 2.  $r_0$ ,  $r_1$ , and  $r_2$  are represented in binary code,  $r_0$  being the LSB, and  $r_2$  the MSB. The code is weighted 1, 2, and 4 from the LSB. Any one of the eight registers can be specified by combinations of this code. See Table 4-4 for the register designation combinations.

When this instruction is executed, one of the registers R0 thru R7 from the register group specified by the PSW RS0 and RS1 bank data is specified. The contents of the specified register is read by the CPU into a temporary register. Then a subsequent decrement ( $-1$ ) by the ALU is followed by a return to the specified register. In this way, the register contents specified by  $r_0$ ,  $r_1$ , and  $r_2$  are decremented.

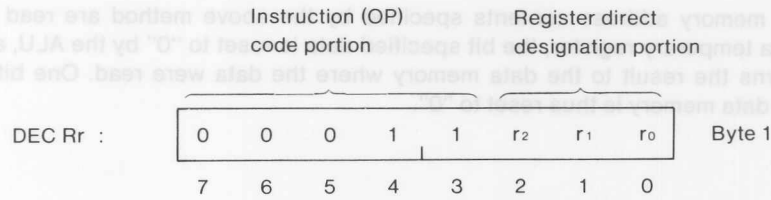


Figure 4-5 DEC Rr bit arrangement

Table 4-4 Register designation table

Register name	$r_2$	$r_1$	$r_0$
Register 0	0	0	0
Register 1	0	0	1
Register 2	0	1	0
Register 3	0	1	1
Register 4	1	0	0
Register 5	1	0	1
Register 6	1	1	0
Register 7	1	1	1

#### 4.3.3 Internal data memory 1-bit data designation

In the MSM80C31/MSM80C51, 1-bit data operations (test, reset, set, complement, transfer) can be executed directly by bit operation instruction only between internal data memory addresses 20H thru 2FH.

The operation of a bit reset instruction is described below as an example.

This instruction (CLR bit address) is a 2-byte 2-machine cycle instruction (see Figure 4-6). The instruction code is indicated in byte 1. Byte 2 is the data memory address and bit designation. The operation bits are specified by the  $b_0$ ,  $b_1$ , and  $b_2$  data in bits 0, 1, and 2 of byte 2. Combinations of this code enable any one of eight bits to be specified. The bit designation combinations are listed in Table 4-5 below.

The data memory is addressed by bits  $b_3$ ,  $b_4$ ,  $b_5$ ,  $b_6$  and  $b_7$  of byte 2. The data memory address is specified by the bits  $b_3$  thru  $b_6$  with  $b_7$  being "0". The bits  $b_3$  -  $b_6$  can be expressed in binary numbers 0 thru 0FH. A total of 16 designations are possible.

When data memory addresses are specified, the data memory bit operation start address 20H is added to the  $b_3$ ,  $b_4$ ,  $b_5$ , and  $b_6$  binary data to obtain the data memory address.

The data memory address contents specified by the above method are read by the CPU into a temporary register, the bit specified data is reset to "0" by the ALU, and the CPU returns the result to the data memory where the data were read. One bit of the specified data memory is thus reset to "0".

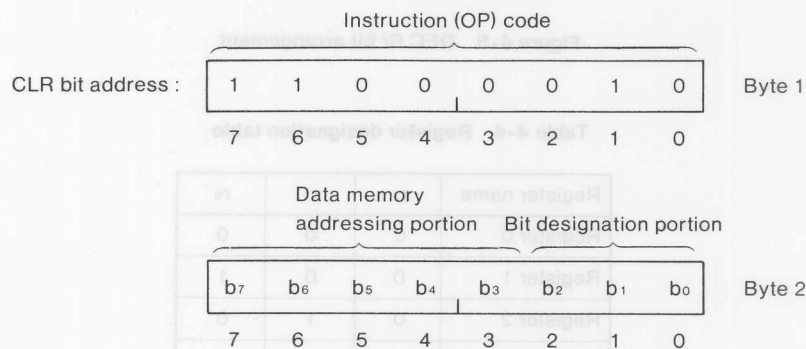


Figure 4-6 CLR bit address bit arrangement

Table 4-5 Bit designation table

Bit name	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>
Bit 0	0	0	0
Bit 1	0	0	1
Bit 2	0	1	0
Bit 3	0	1	1
Bit 4	1	0	0
Bit 5	1	0	1
Bit 6	1	1	0
Bit 7	1	1	1

Table 4-6 Addressing combination table

	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	RAM address	
0	0	0	0	0	0	20H	32
1	0	0	0	0	1	21H	33
2	0	0	0	1	0	22H	34
3	0	0	0	1	1	23H	35
4	0	0	1	0	0	24H	36
5	0	0	1	0	1	25H	37
6	0	0	1	1	0	26H	38
7	0	0	1	1	1	27H	39
8	0	1	0	0	0	28H	40
9	0	1	0	0	1	29H	41
A	0	1	0	1	0	2AH	42
B	0	1	0	1	1	2BH	43
C	0	1	1	0	0	2CH	44
D	0	1	1	0	1	2DH	45
E	0	1	1	1	0	2EH	46
F	0	1	1	1	1	2FH	47

#### 4.4 Special Function Registers (TCON, SCON, . . . ACC, B)

##### 4.4.1 Outline

Special function registers are located in the data address range from 80 to 0FFH. Data addressing of all registers is possible.

Bit addressing can be applied for only the following registers: P0, P1, P2, P3, TCON, SCON, IE, IP, PSW, ACC, and B registers.

If a register which does not exist at a data address is accessed during the use of special function registers, the data read becomes FFH. And if data is written, none of the registers in the CPU are effected at all. Note, however, that since a jump is always made upon execution of a bit test instruction which results in a relative jump at data condition "1", make sure that no instruction is executed for a register which does not exist. A list of special function registers is given in Table 4-7.

Table 4-7 Addressing combination table

RAM address	P0	P1	P2	P3	IE	IP
80H	0	0	0	0	0	0
81H	1	0	0	0	0	0
82H	0	1	0	0	0	0
83H	0	0	1	0	0	0
84H	0	0	0	1	0	0
85H	1	1	0	0	0	0
86H	0	0	1	0	0	0
87H	0	0	0	1	0	0
88H	0	1	1	0	0	0
89H	1	1	1	0	0	0
8AH	0	0	0	1	0	0
8BH	1	0	0	1	0	0
8CH	0	1	0	1	0	0
8DH	1	1	0	1	0	0
8EH	0	0	1	1	0	0
8FH	1	0	1	1	0	0

Table 4-7 List of special function registers

Register name	Bit address								Data address
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	
B	F7	F6	F5	F4	F3	F2	F1	F0	0F0H (240)
ACC	E7	E6	E5	E4	E3	E2	E1	E0	0E0H (224)
PSW	D7	D6	D5	D4	D3	D2	D1	D0	0D0H (208)
IP	BF	BE	BD	BC	BB	BA	B9	B8	0B8H (184)
P3	B7	B6	B5	B4	B3	B2	B1	B0	0B0H (176)
IE	AF	AE	AD	AC	AB	AA	A9	A8	0A8H (168)
P2	A7	A6	A5	A4	A3	A2	A1	A0	0A0H (160)
SBUF									99H (153)
SCON	9F	9E	9D	9C	9B	9A	99	98	98H (152)
P1	97	96	95	94	93	92	91	90	90H (144)
TH1									8DH (141)
TH0									8CH (140)
TL1									8BH (139)
TL0									8AH (138)
TMOD									89H (137)
TCON	8F	8E	8D	8C	8B	8A	89	88	88H (136)
PCON									87H (135)
DPH									83H (131)
DPL									82H (130)
SP									81H (129)
P0	87	86	85	84	83	82	81	80	80H (128)

## 4.4.2 Special Function Registers

## 4.4.2.1 Timer mode register (TMOD)

Name	Address	MSB 7		6	5	4	3	2	1	LSB 0
TMOD	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	
Bit location	Flag	Function								
TMOD.0	M0	M1	M0	Timer/counter 0 mode setting						
		0	0	8-bit timer/counter with 5-bit prescaler						
		0	1	16-bit timer/counter						
		1	0	8-bit timer/counter with 8-bit auto reloading						
TMOD.1	M1	1	1	Timer/counter 0 separated into TL0 (8-bit) timer/counter and TH0 (8-bit) timer/counter. TF0 is set by TL0 carry, and TF1 is set by TH0 carry.						
TMOD.2	C/T	Timer/counter 0 count clock designation control bit. XTAL1 · 2 divided by 12 clocks is the input applied to timer/counter 0 when C/T = "0". The external clock applied to the T0 pin is the input applied to timer/counter 0 when C/T = "1".								
TMOD.3	GATE	When this bit is "0", the TR0 signal is used to control the start and stop of timer/counter 0 counting. If this bit is "1", timer/counter 0 starts counting when both the TR0 bit and INT0 pin are "1", and stops counting when either is changed to "0".								
TMOD.4	M0	M1	M0	Timer/counter 1 mode setting						
		0	0	8-bit timer/counter with 5-bit prescaler						
		0	1	16-bit timer/counter						
TMOD.5	M1	1	0	8-bit timer/counter with 8-bit auto reloading						
		1	1	Timer/counter 1 operation stopped						
TMOD.6	C/T	Timer/counter 1 count clock designation control bit. XTAL1 · 2 divided by 12 clocks is the input applied to timer/counter 1 when C/T = "0". The external clock applied to the T1 pin is the input applied to timer/counter 1 when C/T = "1".								
TMOD.7	GATE	When this bit is "0", the TR1 signal is used to control the start and stop of timer/counter 1 counting. If this bit is "1", timer/counter 1 starts counting when both the TR1 bit and INT1 pin are "1", and stops counting when either is changed to "0".								

## 4.4.2.2 Power control register (PCON)

Name	Address	MSB 7	6	5	4	3	2	1	LSB 0
PCON	87H	SMOD	—	—	—	GF1	GF0	PD	IDL
Bit location	Flag	Function							
PCON.0	IDL	IDLE mode set when this bit is set to "1". CPU operations are stopped when IDLE mode is set, but XTAL1 · 2, timer/counters 0 and 1, the interrupt circuits, and serial ports remain active. IDLE mode is cancelled when the CPU is reset or when an interrupt is generated.							
PCON.1	PD	PD mode set when this bit is set to "1". CPU operations and XTAL1 · 2 are stopped when PD mode is set. PD mode is cancelled only when the CPU is reset.							
PCON.2	GF0	Testing this flag when IDLE mode is cancelled by an interrupt shows whether the interrupt is a normal interrupt or an IDLE mode release interrupt.							
PCON.3	GF1	This is a user's flag which may be used at the user's discretion.							
PCON.4	—	Reserved bit. If the bit is read, the output is a "1".							
PCON.5	—	Reserved bit. If the bit is read, the output is a "1".							
PCON.6	—	Reserved bit. If the bit is read, the output is a "1".							
PCON.7	SMOD	This bit has the following functions when the serial port is in mode 1, 2, or 3. The serial port operation clock is reduced by 1/2 when the bit is "0" for delayed processing. And when the bit is "1", the serial port operation clock is normal for faster processing.							



## 4.4.2.3 Timer control register (TCON)

Name	Address	MSB 7	6	5	4	3	2	1	LSB 0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Bit location	Flag	Function							
TCON.0	IT0	External interrupt 0 signal used in level detect mode when bit is "0", and is used in trigger detect mode when bit is "1".							
TCON.1	IE0	Interrupt flag bit for external interrupt 0. Bit is reset automatically when the interrupt is serviced. Bit can be set and reset by software when IT0 = "1".							
TCON.2	IT1	External interrupt 1 signal used in level detect mode when bit is "0", and is used in trigger detect mode when bit is "1".							
TCON.3	IE1	Interrupt flag bit for external interrupt 1. Bit is reset automatically when the interrupt is serviced. Bit can be set and reset by software when IT1 = "1".							
TCON.4	TR0	Counting start and stop control bit for timer/counter 0. Timer/counter 0 starts counting when this bit is "1", and stops counting when "0".							
TCON.5	TF0	Interrupt flag bit for timer interrupt 0. Bit is reset automatically when the interrupt is serviced. Bit is set to "1" when carry signal is generated from timer/counter 0.							
TCON.6	TR1	Counting start and stop control bit for timer/counter 1. Timer/counter 1 starts counting when this bit is "1", and stops counting when "0".							
TCON.7	TF1	Interrupt flag bit for timer interrupt 1. Bit is reset automatically when the interrupt is serviced. Bit is set to "1" when carry signal is generated from timer/counter 1.							

## 4.4.2.4 Serial port control register (SCON)

Name	Address	MSB 7	6	5	4	3	2	1	LSB 0
SCON	98H	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
Bit location	Flag	Function							
SCON.0	RI	"End of serial port reception" interrupt request flag. This flag must be reset by software during interrupt processing. This flag is set after the eighth bit of data has been received when in mode 0, or by the STOP bit when in any other mode. In mode 2 or 3, however, RI is not set if the RB8 data is "0" with SM2 = "1". RI is set in mode 1 if STOP bit is received when SM2 = "1".							
SCON.1	TI	"End of serial port transmission" interrupt request flag. This flag must be reset by software during interrupt processing. This flag is set after the eighth bit of data has been sent when in mode 0, or after the last bit of data has been sent when in any other mode.							
SCON.2	RB8	The ninth bit of data received in mode 2 or 3 is passed to the RB8. The STOP bit is input to the RB8 if SM2 = "0" when in mode 1. RB8 is not used in mode 0.							
SCON.3	TB8	The TB8 data is sent as the ninth data bit when in mode 2 or 3. Any desired data can be set in TB8 by software.							
SCON.4	REN	Reception enable control bit. No reception when REN = "0". Reception enabled when REN = "1".							
SCON.5	SM2	If the ninth bit of data is "0" with SM2 = "1" in mode 2 or 3, the "end of reception" signal is not set into the RI bit. Nor is the "end of reception" signal set into the RI bit if the STOP bit is not "1" with SM2 = "1" in mode 1.							
SCON.6	SM1	SM0	SM1	MODE					
		0	0	0	8-bit shift register I/O				
		0	1	1	8-bit UART variable baud rate				
SCON.7	SM0	1	0	2	9-bit UART 1/32 XTAL1, 1/64 XTAL1 baud rate				
		1	1	3	9-bit UART variable baud rate				

## 4.4.2.5 Interrupt enable register (IE)

Name	Address	MSB 7	6	5	4	3	2	1	LSB 0
IE	A8H	EA	—	—	ES	ET1	EX1	ET0	EX0
Bit location	Flag	Function							
IE.0	EX0	Interrupt control bit for external interrupt 0. Interrupt disabled when bit is "0". Interrupt enabled when bit is "1".							
IE.1	ET0	Interrupt control bit for timer interrupt 0. Interrupt disabled when bit is "0". Interrupt enabled when bit is "1".							
IE.2	EX1	Interrupt control bit for external interrupt 1. Interrupt disabled when bit is "0". Interrupt enabled when bit is "1".							
IE.3	ET1	Interrupt control bit for timer interrupt 1. Interrupt disabled when bit is "0". Interrupt enabled when bit is "1".							
IE.4	ES	Interrupt control bit for serial port. Interrupt disabled when bit is "0". Interrupt enabled when bit is "1".							
IE.5	—	Reserved bit. If the bit is read out, "1" is read.							
IE.6	—	Reserved bit. If the bit is read out, "1" is read.							
IE.7	EA	Overall interrupt control bit. All interrupts disabled when bit is "0". All interrupts enabled when bit is "1".							

## 4.4.2.6 Interrupt priority register (IP)

Name	Address	MSB 7	6	5	4	3	2	1	LSB 0
IP	B8H	—	—	—	PS	PT1	PX1	PT0	PX0
Bit location	Flag	Function							
IP.0	PX0	Interrupt priority bit for external interrupt 0. Priority assigned when bit is "1".							
IP.1	PT0	Interrupt priority bit for timer interrupt 0. Priority assigned when bit is "1".							
IP.2	PX1	Interrupt priority bit for external interrupt 1. Priority assigned when bit is "1".							
IP.3	PT1	Interrupt priority bit for timer interrupt 1. Priority assigned when bit is "1".							
IP.4	PS	Interrupt priority bit for serial port. Priority assigned when bit is "1".							
IP.5	—	Reserved bit. If the bit is read out, "1" is read.							
IP.6	—	Reserved bit. If the bit is read out, "1" is read.							
IP.7	—	Reserved bit. If the bit is read out, "1" is read.							

## 4.4.2.7 Program status word register (PSW)

Name	Address	MSB 7	6	5	4	3	2	1	LSB 0
PSW	D0H	CY	AC	F0	RS1	RS0	OV	F1	P
Bit location	Flag	Function							
PSW.0	P	Accumulator (ACC) parity indicator. “1” when the number of “1” in the accumulator is an odd number, and “0” when an even number.							
PSW.1	F1	User flag which may be set to “0” or “1” as desired by the user.							
PSW.2	OV	Overflow flag which is set if the carry C6 from bit 6 of the ALU or CY is “1” as a result of an arithmetic operation. The flag is also set to “1” if the resultant product of a multiplication instruction (MULA, B) is greater than 00FFH, and is set to “0” if the product is equal or less than 00FFH.							
PSW.3	RS0	RAM register bank switch							
		RS1	RS0	BANK	RAM ADDRESS				
		0	0	0	00H – 07H				
PSW.4	RS1	0	1	1	08H – 0FH				
		1	0	2	10H – 17H				
		1	1	3	18H – 1FH				
PSW.5	F0	User flag which may be set to “0” or “1” as desired by the user.							
PSW.6	AC	Auxiliary carry flag. This flag is set to “1” if a carry C <sub>3</sub> is generated from bit 3 of the ALU as a result of executing an arithmetic operation instruction. In all other cases, the flag is reset to “0”.							
PSW.7	CY	Main carry flag. This flag is set to “1” if a carry C <sub>7</sub> is generated from bit 7 of the ALU as a result of execution of an arithmetic operation instruction. In all other cases, the flag is reset to “0”.							

## 4.5 Timer/Counters 0 and 1

### 4.5.1 Outline

Timer/counters 0 and 1 are both capable of independent 16-bit binary counting. All control of timer/counters 0 and 1 is handled by the timer control register (TCON) and the timer mode register (TMOD). Both timer/counter 0 and 1 can be set independently to modes 0 thru 3 for a diversity of applications.

### 4.5.2 Timer/counter 0 and 1 counting control

The start and stop of counting in timer/counters 0 and 1 is controlled by the bit-4, TR0, and bit-6, TR1, in the timer control register (TCON 88H) as indicated in Table 4-8. TR0 controls timer/counter 0, and TR1 controls timer/counter 1. Timer/counter operation is stopped when the bit data is "0", and enabled when "1". The overall control circuit for timer/counters 0 and 1 is shown in Figure 4-7 (excluding timer mode 3).

Table 4-8 Timer control register (TCON 88H)

	Timer 1		Timer 0					
Bit	7	6	5	4	3	2	1	0
Flag	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Set		•		•				



**4.5.3 Timer/counter 0 and 1 count clock designation**

Designation of count clock inputs to timer/counters 0 and 1 is controlled by the bit 2 and 6, C/T, in the timer mode register (TMOD 89H).

Timer/counter 0 is controlled by the bit 2, C/T, and timer/counter 1 is controlled by the bit 6, C/T.

The internal clock is applied to the timer/counter when the C/T bit is "0". This internal clock is the result of dividing  $XTAL1 \cdot 2$  by 12. The S3 timing signal becomes the clock.

The external clock is applied to the timer/counter when the C/T bit is "1". The external clock applied to the T0 pin serves as the timer/counter 0 input, while the external clock applied to the T1 pin serves as the timer/counter 1 input.

**Table 4-9 Timer mode register (TMOD 89H)**

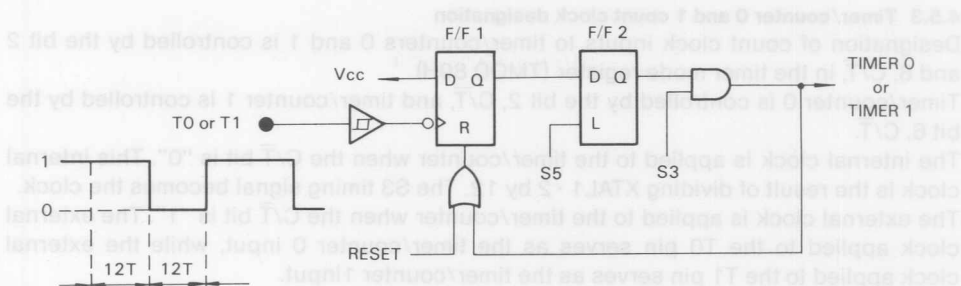
Bit	Timer 1				Timer 0			
	7	6	5	4	3	2	1	0
Flag	GATE	C/T	M1	M0	GATE	C/T	M1	M0
Enable count		•				•		

**4.5.3.1 External clock detector circuit for timer/counters 0 and 1**

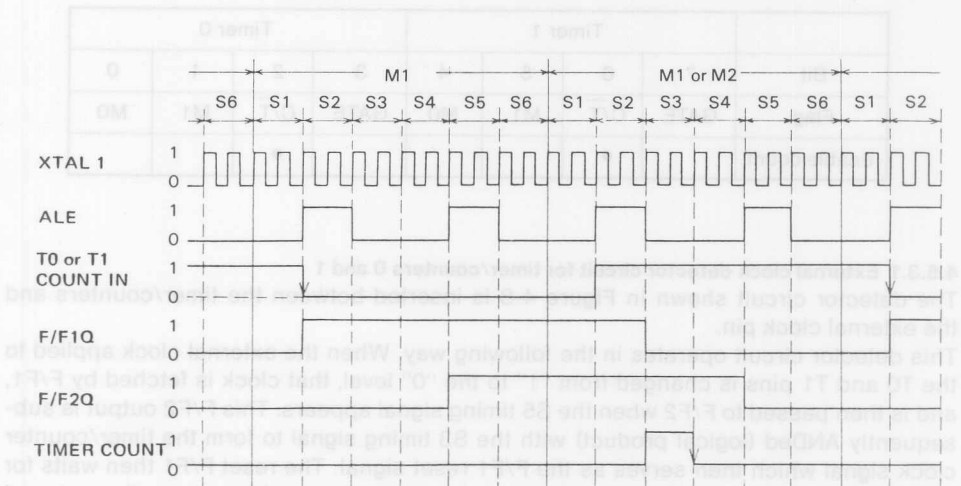
The detector circuit shown in Figure 4-8 is inserted between the timer/counters and the external clock pin.

This detector circuit operates in the following way. When the external clock applied to the T0 and T1 pins is changed from "1" to the "0" level, that clock is fetched by F/F1, and is then passed to F/F2 when the S5 timing signal appears. This F/F2 output is subsequently ANDed (logical product) with the S3 timing signal to form the timer/counter clock signal which then serves as the F/F1 reset signal. The reset F/F1 then waits for the next external clock. The "0" and "1" signal cycle widths of the respective external clocks applied to the T0 and T1 pins must be minimal and a period 12 times (12T) to  $XTAL1 \cdot 2$  oscillator clock cycle T is required. The operational time chart for this detector circuit is outlined in Figure 4-9.





**Figure 4-8 T0 and T1 external clock detector circuit**



**Figure 4-9** Detector circuit operational time chart

#### 4.5.4 Counting control of timer/counters 0 and 1 by $\overline{\text{INT}}$ pin

In addition to control by timer control register, TCON TR0 and TR1, timer/counter 0 and 1 counting start and stop can also be controlled by the signal level applied to the external interrupt pin. The GATE data values of bits 3 and 7 in the timer mode register (TMOD 89H) indicated in Table 4-10 are used to control this.

Timer/counter 0 is controlled by the bit 3 GATE bit. When the GATE bit is "0", counting of timer/counter 0 is started and stopped only by TR0.

When the GATE bit is "1", counting timer/counter 0 is enabled if the TR0 signal and the signal applied to the  $\overline{\text{INT}}$  pin are both "1". Counting is subsequently stopped if either of the signals is changed to "0" level.

Timer/counter 1 is controlled by the bit-7 GATE bit. The functional operation is the same as for timer/counter 0. The GATE -  $\overline{\text{INT}}$  timer/counter counting control circuit is outlined in Figure 4-10, and the control table is given in Table 4-11.

Table 4-10 Timer mode register (TMOD 89H)

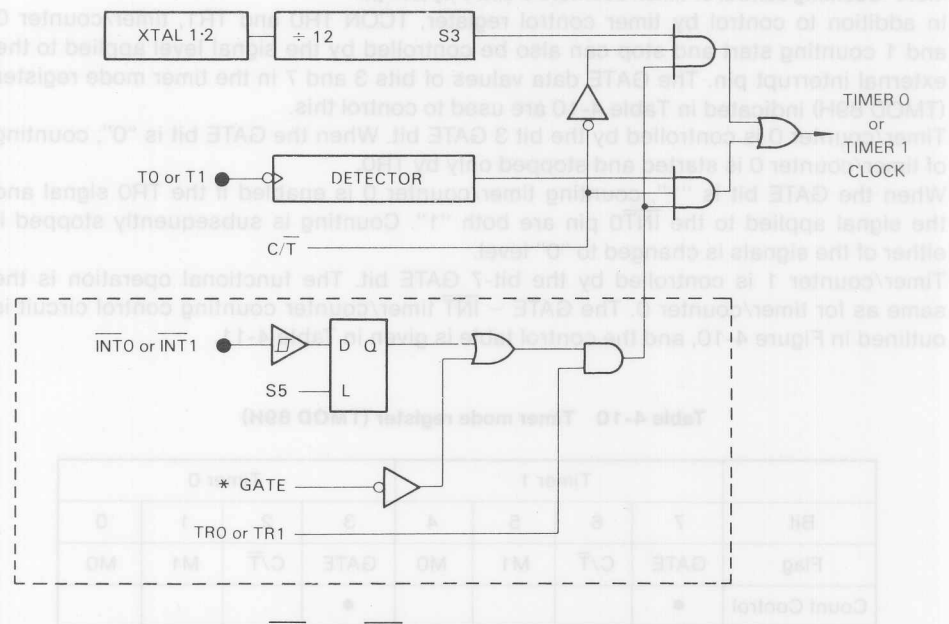
Bit	Timer 1				Timer 0			
	7	6	5	4	3	2	1	0
Flag	GATE	C/ $\overline{\text{T}}$	M1	M0	GATE	C/ $\overline{\text{T}}$	M1	M0
Count Control	•				•			

4

TIMER 1				
GATE	0	0	0	0
TR1	0	1	0	1
INT1	×	×	0	1
RUN	•			
STOP	•	•		

TIMER 0				
GATE	0	0	1	1
TR0	0	1	0	1
INT0	×	×	0	1
RUN	•			
STOP	•	•		

Note: • implies timer/counter state.

Figure 4-10  $\overline{\text{INT0}}$  and  $\overline{\text{INT1}}$  timer/counter start/stop control circuitTable 4-11  $\text{GATE} \cdot \overline{\text{INT}} \cdot \text{TR}$  timer/counter control tables

	TIMER 0				
GATE	0	0	1	1	1
TR0	0	1	0	1	1
$\overline{\text{INT0}}$	×	×	0	0	1
RUN		●			●
STOP	●		●	●	

	TIMER 1				
GATE	0	0	1	1	1
TR1	0	1	0	1	1
$\overline{\text{INT1}}$	×	×	0	0	1
RUN		●			●
STOP	●		●	●	

Note: ● implies timer/counter state.

#### 4.5.5 Timer/counters 0/1 and timer modes

##### 4.5.5.1 Outline

The timer/counter 0 and 1 timer modes are set by combinations of M0 and M1 bit data in the timer mode register (TMOD 89H). The timer modes which can be set are 0, 1, 2, and 3.

Timer/counter 0 modes are specified by M0 and M1 of bits 0 and 1, and timer/counter 1 modes are specified by M0 and M1 of bits 4 and 5.

Table 4-12 Timer mode register (TMOD 89H)

	TIMER COUNTER 1				TIMER COUNTER 0			
Bit	7	6	5	4	3	2	1	0
Flag	GATE	C/T	M1	M0	GATE	C/T	M1	M0
Mode set			•	•			•	•

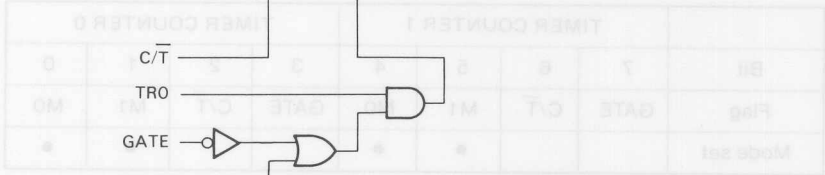
##### 4.5.5.2 Mode 0

In mode 0, timer/counters 0 and 1 both become 13-bit timer/counters by the circuit connection shown in Figures 4-11 and 4-12. TL0 and TL1 in timer/counters 0 and 1 serve as the counter for the five lower bits, and TH0 and TH1 serve as the counter for the eight upper bits.

Timer flag 0 is set by the timer/counter 0 carry signal, and timer flag 1 is set by the timer/counter 1 carry signal. Timer/counter 1 carry signal can be used as the serial port clock source.

Although the three upper bits of TL0 and TL1 are operative, they are invalid as signals.

M1	M0
0	0



**Figure 4-11 Timer/counter 0 mode 0**

```
graph LR; XTAL1[XTAL 1] --> Div12[÷ 12]; Div12 -- S3 --> DETECTOR[DETECTOR]; DETECTOR --> TF1[TF1]
```

**4.5.5.3 Mode 1**

In mode 1, timer/counters 0 and 1 both become 16-bit timer/counters by the circuit connection shown in Figures 4-13 and 4-14.

TL0 and TL1 in timer/counters 0 and 1 serve as the counter for the eight lower bits, and TH0 and TH1 serve as the counter for the eight upper bits.

Timer flag 0 is set by the timer/counter 0 carry signal, and timer flag 1 is set by the timer/counter 1 carry signal. Timer/counter 1 carry signal can be used as the serial port clock source.

M1	M0
0	1

Figure 4-13 Timer/counter 0 mode 1

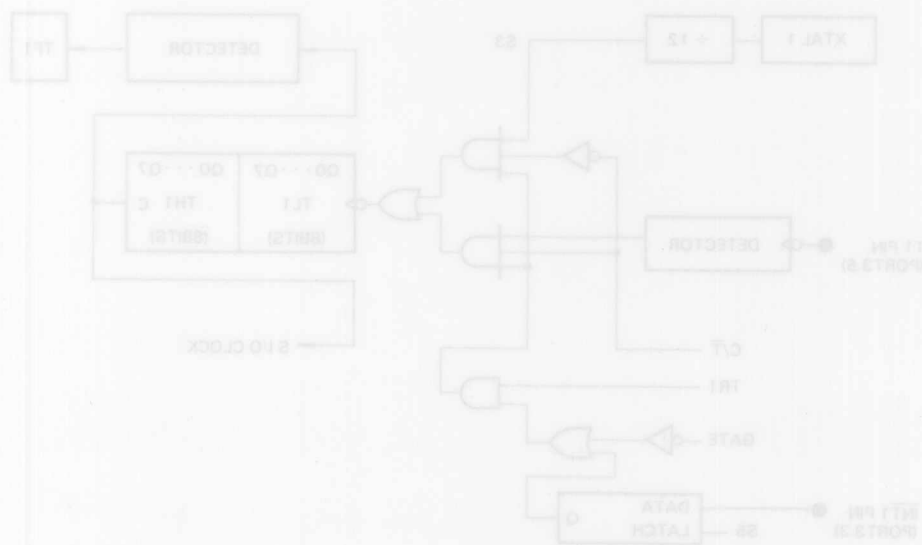


Figure 4-14 Timer/counter 1 mode 1

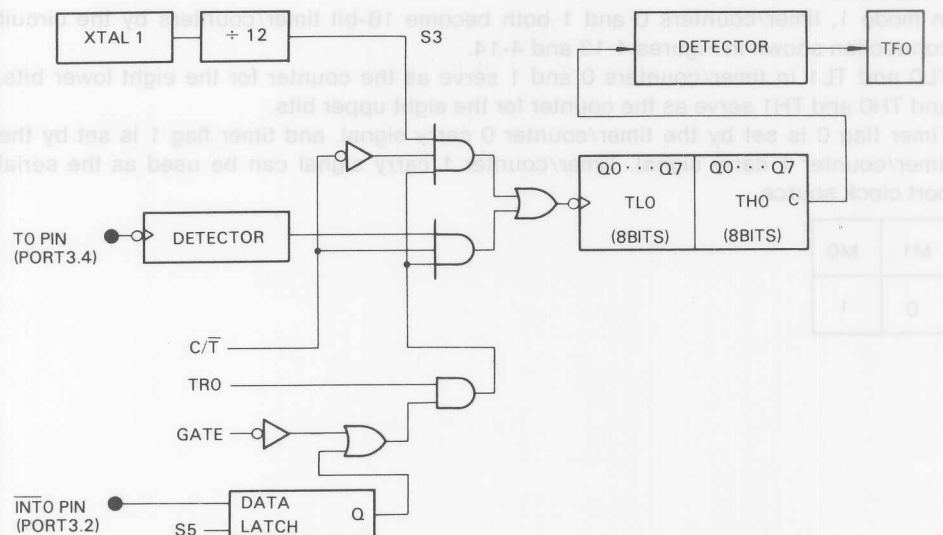


Figure 4-13 Timer/counter 0 mode 1

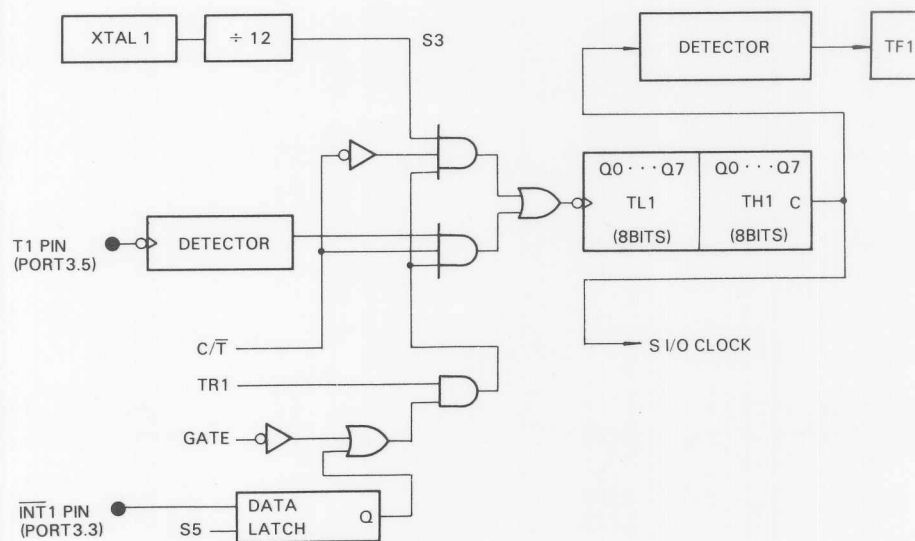


Figure 4-14 Timer/counter 1 mode 1

#### 4.5.5.4 Mode 2

In mode 2, timer/counters 0 and 1 both become 8-bit timer/counters with 8-bit auto reloader registers by the circuit connection shown in Figures 4-15 and 4-16.

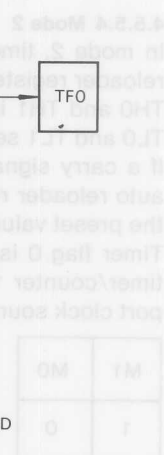
TH0 and TH1 in timer/counters 0 and 1 serve as the 8-bit auto reloader section, and TL0 and TL1 serve as the timer/counter section.

If a carry signal is generated by the 8-bit timer/counter TLO and TL1, the respective auto reloader register data is preset in the timer/counter, and counting proceeds from the preset value.

Timer flag 0 is set by the timer/counter 0 carry signal, and timer flag 1 is set by the timer/counter 1 carry signal. Timer/counter 1 carry signal can be used as the serial port clock source.

M1	MO
1	0





## 5 Timer/counter 0 mode 2



## 6 Timer/counter 1 mode 2

**4.5.5.5 Mode 3**

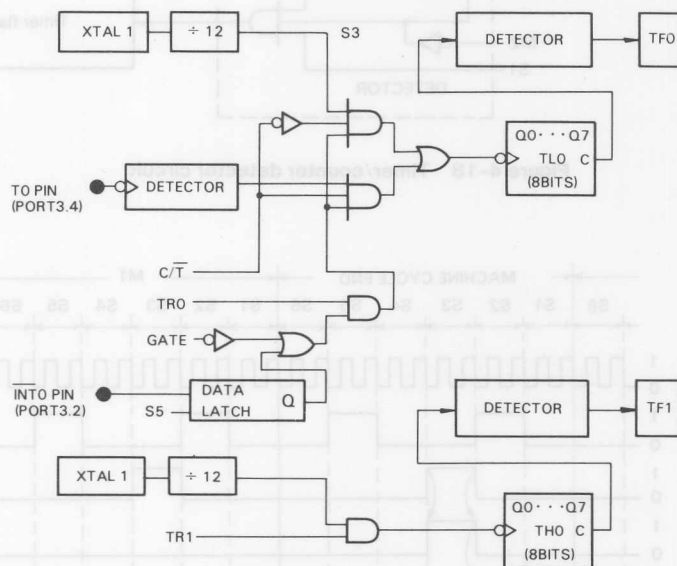
In mode 3, timer/counter 0 TLO and TH0 become independent 8-bit timer/counters by the circuit connection shown in Figure 4-17. Timer/counter 1 does not operate when mode 3 is set. The TLO 8-bit timer/counter is controlled in the same way as the regular timer/counter 0, timer flag 0 being set if a carry signal is generated from TLO.

The TH0 8-bit timer/counter is controlled only by TR1, and the control covers only the start and stop of the counting.

Timer flag 1 is set by a carry signal generated from TH0.

When timer/counter 0 is set to mode 3, timer/counter 1 can operate in modes 0, 1, or 2, and be used as the serial port clock source. Control of timer/counter 1 counting start and stop in this case is handled between the operating mode and mode 3. If mode 3 is set, the timer/counter 1 counting operation is stopped.

M1	M0
1	1



**Figure 4-17** Timer/counter 0 mode 3

#### 4.5.6 Timer/counter carry signal detector circuit

The detector circuit shown in Figure 4-18 is inserted between the MSM80C31/MSM80C51 timer/counter carry output and the timer flag. The purpose of this detector is to prevent loss of the timer flag by operation data when execution of the OR, AND, EOR, RESET bit, or MOV bit instruction which destination operand is the timer control register is completed. If a timer carry is generated during execution of one of these instructions on the timer control register (TCON), the timer flag may be lost. Hence, even if a timer carry signal is generated during execution of an instruction, that flag will not be set while the instruction is still being executed. The flag is set at  $M2 \cdot S1$  during execution of the next instruction. If a timer carry is generated during M1 thru M3 when executing a 4-machine cycles instruction, the timer flag is set during M3 or M4. See Figure 4-19 for the time chart.

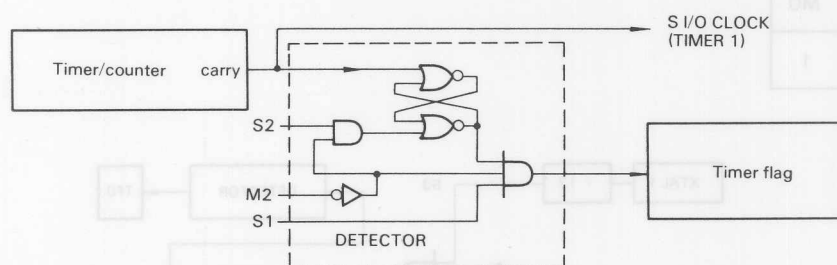


Figure 4-18 Timer/counter detector circuit

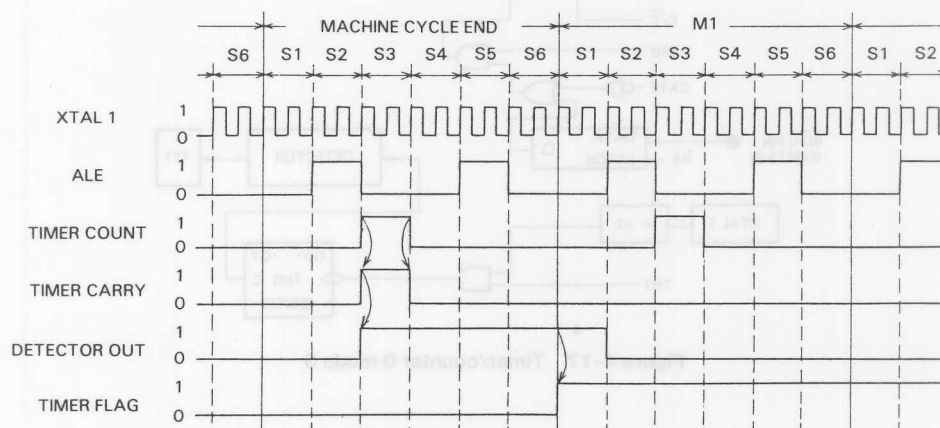


Figure 4-19 Timer flag setting time chart



## 4.6.2 Special function registers for serial port

### 4.6.2.1 SCON (Serial Port Control Register)

SCON is an 8-bit special function register consisting of control bits for specifying serial port operation modes and enabling/disabling data reception, storage bits for the 9th bit of data sent and received during 11-bit frame UART mode, and the serial port status flag.

In addition to specifying SCON by data address 98H, each bit can be specified individually.

The functions of each SCON bit are listed in Table 4-13, and the functions of each operational mode specified by SCON are indicated in Table 4-14.

Table 4-13 SCON

Bit	Symbol	Function
0	RI	"End of reception" flag. This is the interrupt request flag set by hardware when reception of one frame has been completed. The interrupt is generated by ORing with the TI flag. Since the flag cannot be cleared by hardware, it must be cleared by software.
1	TI	"End of transmission" flag. This is the interrupt request flag set by hardware when transmission of one frame has been completed. The interrupt is generated by ORing with the RI flag. Since the flag cannot be cleared by hardware, it must be cleared by software.
2	RB8	Storage of the 9th bit of the data received during 11-bit frame UART mode (mode 2 or 3). When in 10-bit frame UART mode (mode 1), the stop bit is stored, instead.
3	TB8	Storage of the 9th bit of the data to be sent during 11-bit frame UART mode (mode 2 or 3).
4	REN	Receive enable bit. Reception is not activated if REN is not set.
5	SM2	If SM2 is set when in 11-bit frame UART mode (mode 2 or 3), and the 9th bit of the received data is "1", the received data is accepted and loaded into SBUF and RB8, and the RI flag is set. If the 9th bit of the received data is "0", the received data is disregarded and the SBUF, RB8, and RI flags remain unchanged. This function is used to enable communication between processors in multi-processor systems. If SM2 is set when in 10-bit frame UART mode (mode 1) and the stop bit "1" cannot be received, the received data is disregarded, and the SBUF, RB8, and RI flags remain unchanged. When SM2 = "0", however, data is received irrespective of the "0"/"1" status of the stop bit. SM2 must be cleared when in I/O extension mode (mode 0).
6	SM1	Used in setting serial port operation mode. See Table 4-14.
7	SM0	Used in setting serial port operation mode. See Table 4-14.

Table 4-14 Serial port operation modes

SM0	SM1	Mode	Function	Baud rate
0	0	0	I/O extension	$1/12 F_{osc}$
0	1	1	10-bit frame UART	Variable
1	0	2	11-bit frame UART	$1/32 F_{osc}$ or $1/64 F_{osc}$
1	1	3	11-bit frame UART	Variable

Note:  $F_{osc}$  denotes frequency of fundamental oscillator ( $XTAL1 \cdot 2$ )

#### 4.6.2.2 SBUF (serial port buffer register)

SBUF is an 8-bit special function register used to store sending and receiving data. Although the SBUF is specified by the same data address 99H for both writing and reading, physically separate registers are specified. That is, the sending circuit SBUF is specified by instructions where SBUF is used as a destination operand, and the receiving circuit SBUF is specified by instructions where SBUF is used as a source operand.

#### 4.6.2.3 SMOD (double baud rate bit)

SMOD controls the division of the baud rate clock source when the serial port is in UART mode (mode 1, 2, or 3).

If SMOD is cleared to "0" when in mode 1 or 3, the timer/ counter 1 overflow frequency divided by 2 becomes the baud rate clock source. And if SMOD is set to "1", the timer/counter 1 overflow frequency becomes the baud rate clock source.

If SMOD is cleared to "0" when in mode 2,  $1/2XTAL1 \cdot 2$  divided by 2 becomes the baud rate clock source. And if SMOD is set to "1",  $1/2XTAL1 \cdot 2$  becomes the baud rate clock source.

SMOD corresponds to bit 7 of PCON (Power Control Register) specified by data address 87H. Designation by bit address is not possible.

#### 4.6.3 Operating modes

##### 4.6.3.1 Mode 0

###### 4.6.3.1.1 Outline

Mode 0 is the I/O extension mode where input and output of 8-bit data via RXD (P3.0) is enabled synchronized with the output clock from RXD (P3.1).

The baud rate in mode 0 is fixed to 1/12 of the fundamental oscillator ( $XTAL1 \cdot 2$ ) frequency to enable the serial port to operate synchronized with the basic MSM80C31/MSM80C51 timing.

A block diagram of the mode 0 serial port is shown in Figure 4-21, the operational timing chart is shown in Figure 4-22, and the serial port operation timing in relation to the basic MSM80C31/MSM80C51 timing is shown in Figure 4-23.

###### 4.6.3.1.2 Mode 0 baud rate

In mode 0, the baud rate is determined by the following equation to synchronize operations with the basic MSM80C31 /MSM80C51 timing.

$$B = F_{osc} \times \frac{1}{12}$$

where B is baud rate, and  $F_{osc}$  is the fundamental ( $XTAL1 \cdot 2$ ) frequency.

###### 4.6.3.1.3 Mode 0 output operation

Data output is commenced by writing data in SBUF.

The SBUF data is obtained sequentially from RXD one machine cycle after completion of the SBUF data write instruction, the LSB appearing first.

Two states after starting the LSB output, output of the TXD synchronizing clock is commenced. This synchronized clock is at level "0" from the latter half of S3 thru to the first half of S6, and at "1" level from the latter half of S6 thru to the first half of S3. The transmit circuit is initialized immediately following completion of output of the MSB, and the TI flag is set at the first  $M1 \cdot S3$  cycle after that.

###### 4.6.3.1.4 Mode 0 input operation

Data input is initiated when  $REN = "1"$  and  $RI = "0"$ . This is achieved by an instruction used to set REN or by an instruction used to clear the RI flag or by an instruction which does both simultaneously.

Output of the TXD synchronizing clock is initiated nine states after  $REN = "1"$  and  $RI = "0"$ . The synchronized clock is at level "0" from the latter half of S3 thru to the first half of S6, and at level "1" from the latter half of S6 thru to the first half of S3.

The RXD data is read sequentially into an input shift register in the serial port just before the synchronized clock is changed from "0" to "1".

When input of the 8-bit data is completed, loading of the input shift register data into SBUF (with the LSB at the beginning of the input data) occurs at the same time that the receiving circuit is initialized. The RI flag is then set at the first  $M1 \cdot S3$  cycle after completion of input of the 8-bit data.

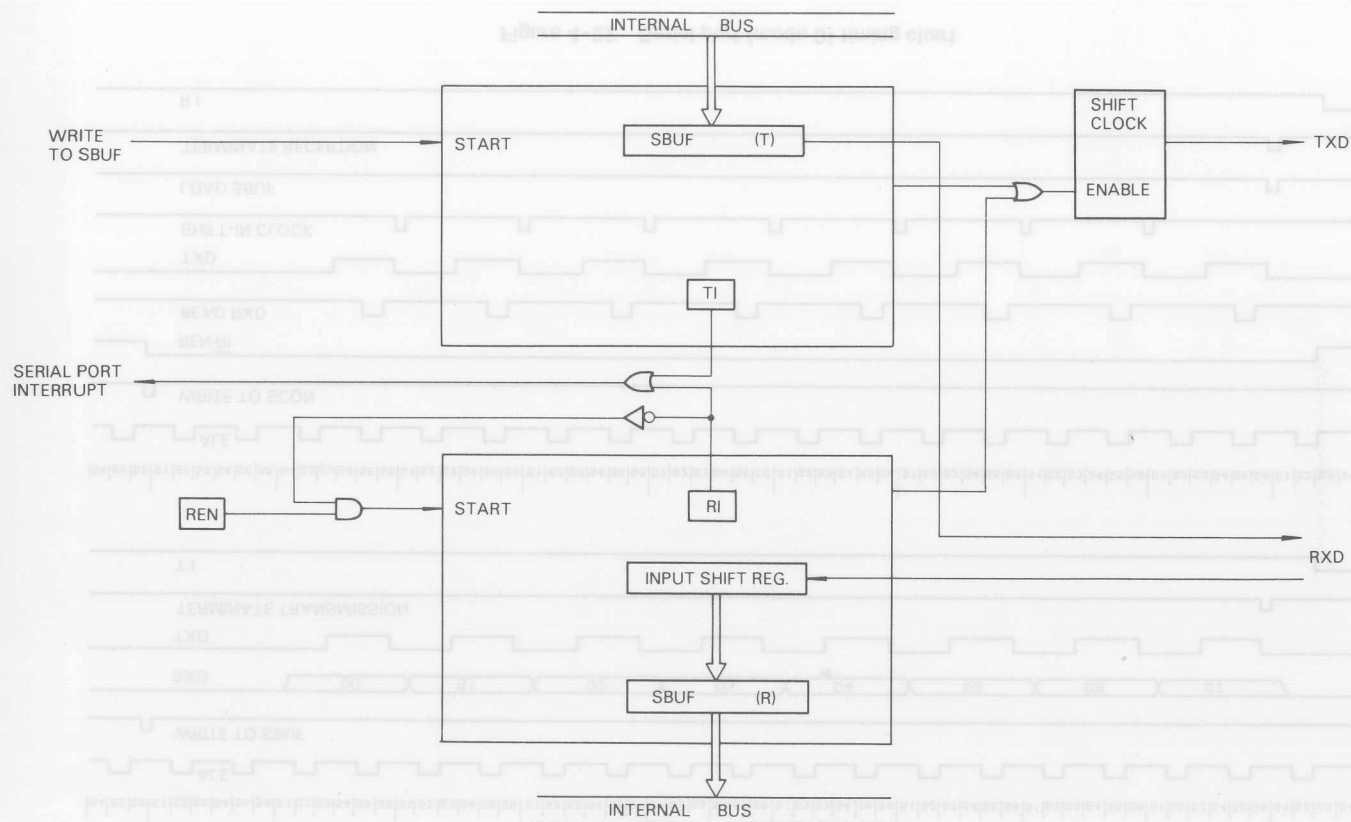


Figure 4-21 Serial port (mode 0)



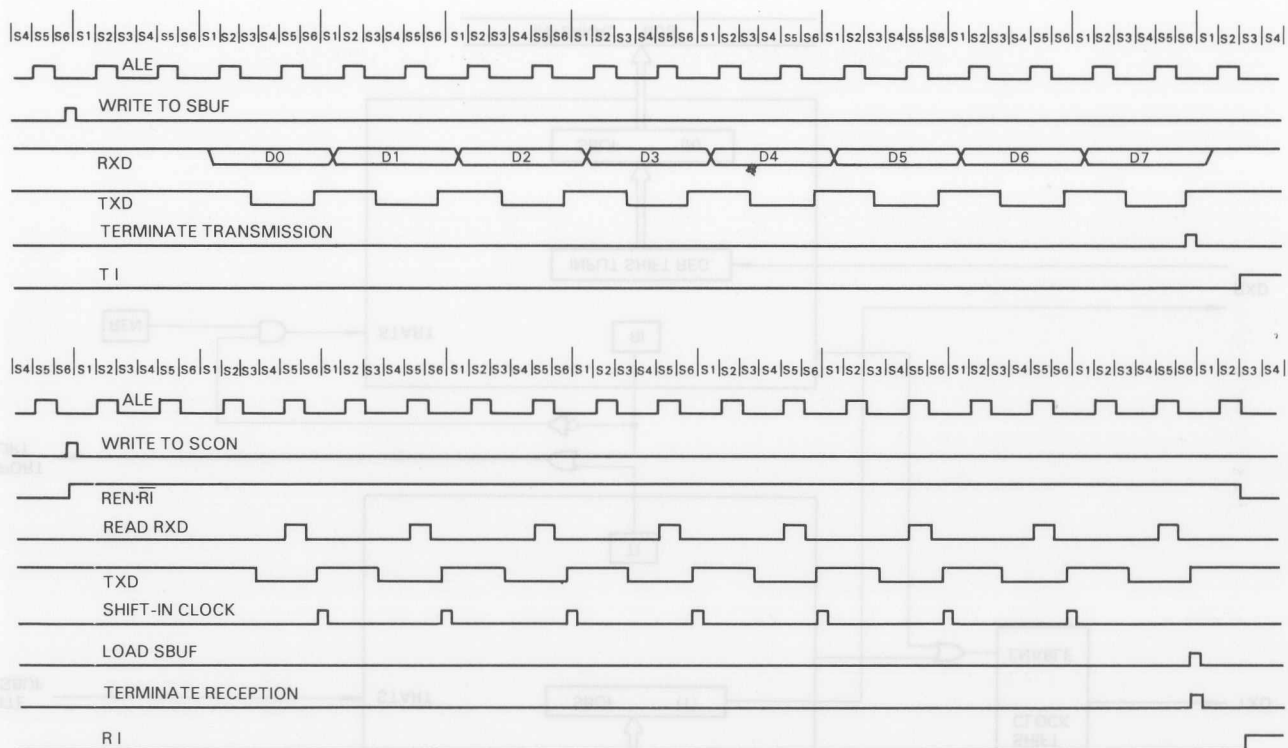


Figure 4-22 Serial port (mode 0) timing chart

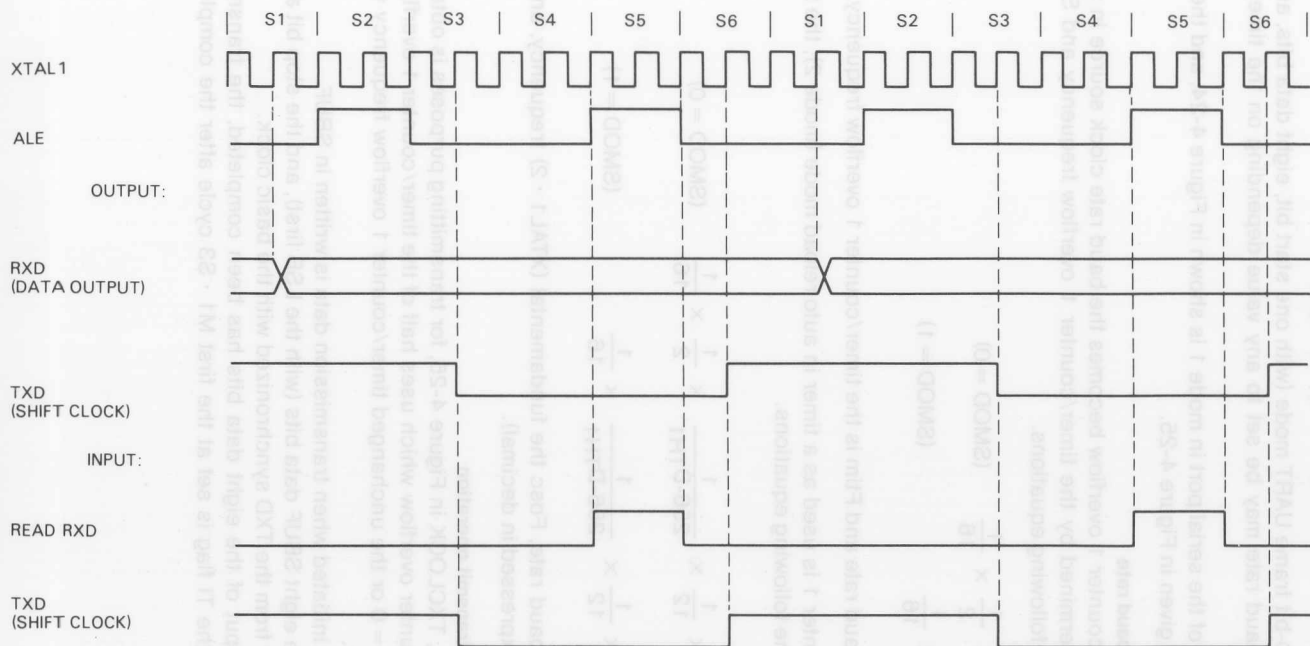


Figure 4-23 Serial port (mode 0) timing and corresponding basic timing

#### 4.6.3.2 Mode 1

##### 4.6.3.2.1 Outline

Mode 1 is the 10-bit frame UART mode (with one start bit, eight data bits, and one stop bit) where the baud rate may be set to any value depending on the timer/counter 1 setting.

A block diagram of the serial port in mode 1 is shown in Figure 4-24, and the operational timing chart is given in Figure 4-25.

##### 4.6.3.2.2 Mode 1 baud rate

Since the timer/counter 1 overflow becomes the baud rate clock source in mode 1, the baud rate is determined by the timer/counter 1 overflow frequency and SMOD value according to the following equations.

$$B = F_{tim} \times \frac{1}{2} \times \frac{1}{16} \quad (SMOD = 0)$$

$$B = F_{tim} \times \frac{1}{16} \quad (SMOD = 1)$$

where B is the baud rate and  $F_{tim}$  is the timer/counter 1 overflow frequency.

When timer/counter 1 is used as a timer in auto reload mode (mode 2), the baud rate is determined by the following equations.

$$B = F_{osc} \times \frac{1}{12} \times \frac{1}{256 - D_{TH1}} \times \frac{1}{2} \times \frac{1}{16} \quad (SMOD = 0)$$

$$B = F_{osc} \times \frac{1}{12} \times \frac{1}{256 - D_{TH1}} \times \frac{1}{16} \quad (SMOD = 1)$$

where B is the baud rate,  $F_{osc}$  the fundamental ( $XTAL1 \cdot 2$ ) frequency, and  $D_{TH1}$  the TH1 contents (expressed in decimal).

##### 4.6.3.2.3 Mode 1 transmit operation

The basic clock, TXCLOCK in Figure 4-25, for transmitting purposes is obtained from a hexadecimal counter overflow which uses half of the timer/counter 1 overflow frequency when  $SMOD = 0$  or the unchanged timer/counter 1 overflow frequency when  $SMOD = 1$  as the clock.

Transmission is initiated when transmission data is written in SBUF.

The start bit, the eight SBUF data bits (with the LSB first), and the stop bit are transmitted sequentially from the TXD synchronized with the basic clock.

As soon as output of the eight data bits has been completed, the transmit circuit is initialized, and the TI flag is set at the first  $M1 \cdot S3$  cycle after the completion of that output.

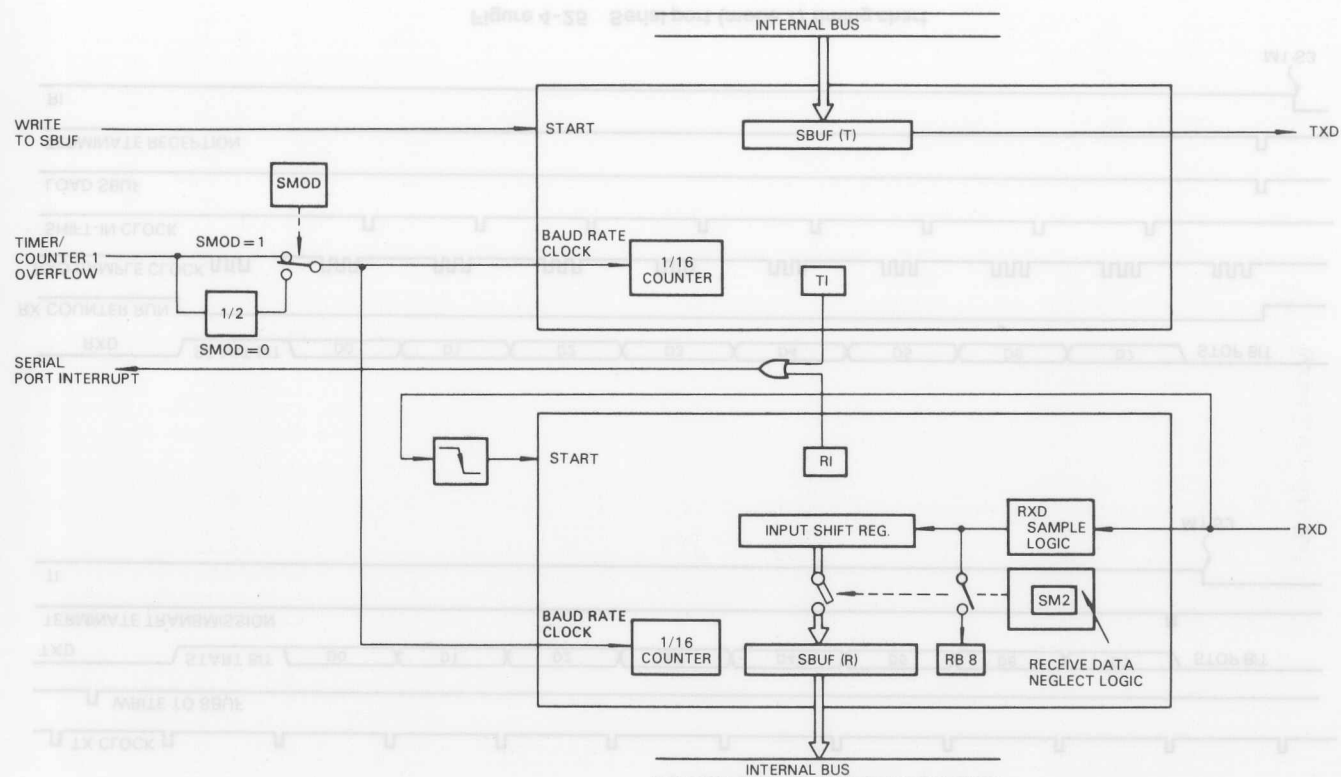


Figure 4-24 Serial port (mode 1)

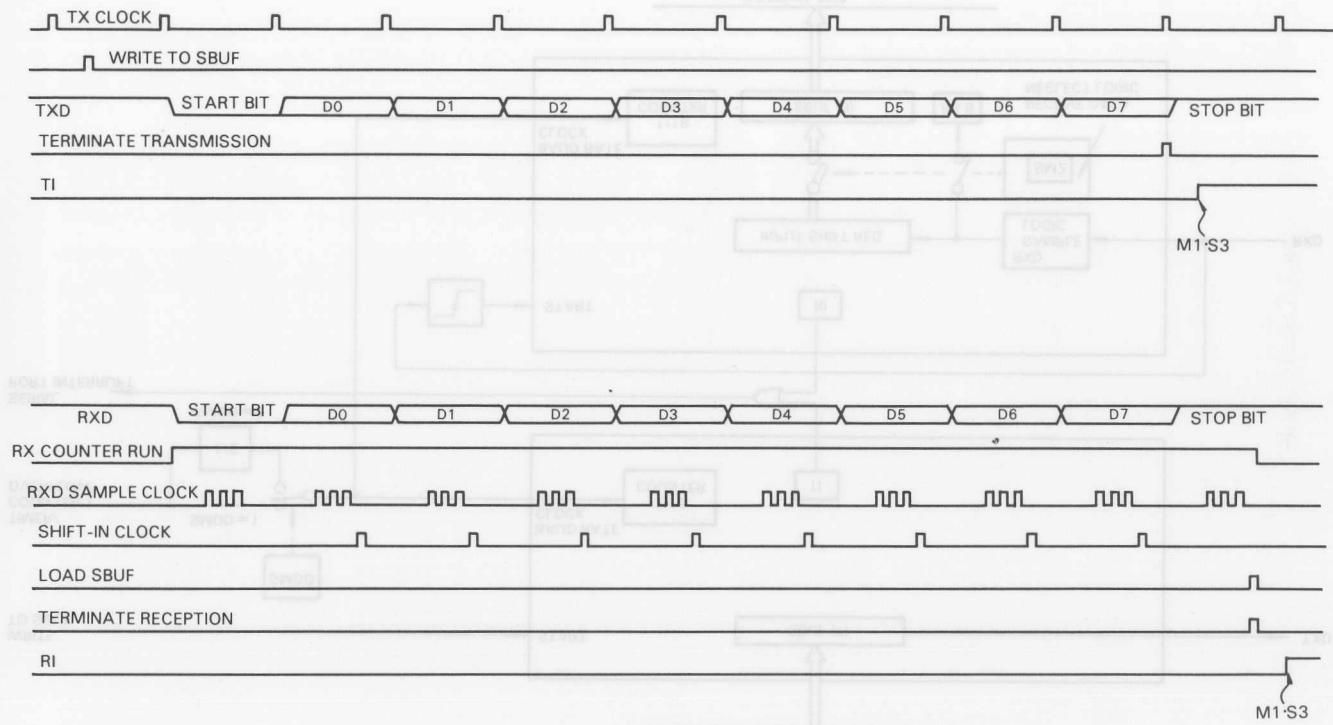


Figure 4-25 Serial port (mode 1) timing chart

**4.6.3.2.4 Mode 1 receive operation**

The receive circuit timing is generated by a hexadecimal counter which employs the halved timer/counter 1 overflow frequency (when SMOD = 0) or the unchanged timer/counter 1 overflow frequency (when SMOD = 1) as the clock, and the input data received from the RXD is bit synchronized. That is, at the same time that reception is started, following input of the start bit, the hexadecimal counter commences to count up, and with one complete round of the hexadecimal counter corresponding to one bit of received data, reception is continued by the receive circuit. Therefore, timer/counter 1 must be set so that the period of a single round of the hexadecimal counter is equal to the reception data baud rate.

The RXD change from "1" to "0" is regarded as the beginning of the start bit for reception.

When this "1" to "0" change in RXD is detected, the hexadecimal counter which had been reset starts to count up. When the hexadecimal counter is in state 7, 8, and 9, the start bit is sampled, and is accepted as valid if at least two of the three sampled values are "0", thereby enabling data reception to continue. If two or three of the sampled values are "1", the start bit becomes invalid, and the receive circuit is initialized when the hexadecimal counter reaches state 10.

The reception data is sampled when the hexadecimal counter is in state 7, 8, and 9, and the more common value of the three sampled values is read sequentially as data into the input shift register.

If the conditions stated below are satisfied when the hexadecimal counter is in state 10 during the period of the stop bit, the input shift register data (the LSB being read first) is loaded into SBUF, and the sampled stop bit is read into RB8, thereby initializing the receive circuit.

- Conditions: (1) RI = "0"  
 (2) SM2 = "0", or  
 SM2 = "1" and sampled stop bit = "1"

The RI flag is set at the first M1 · S3 cycle after the initialization.

If the above conditions are not satisfied, the received data is disregarded, and the receive circuit is initialized without change to the SBUF, RB8, and RI flags.

Since the receive circuit is divided into two stages (input shift register and SBUF), processing of the previous receive data may be completed within the interval up to the stop bit period of the next frame.

#### 4.6.3.3 Mode 2

##### 4.6.3.3.1 Outline

Mode 2 is an 11-bit frame UART mode (with one start bit, eight data bits, one multi-purpose bit, and one stop bit) where the baud rate is 1/64th or 1/32nd of the fundamental oscillator ( $XTAL1 \cdot 2$ ) frequency. A block diagram of the serial port in mode 2 is shown in Figure 4-26, and the operational timing chart is given in Figure 4-27.

##### 4.6.3.3.2 Mode 2 baud rate

Since the fundamental oscillator frequency divided by two serves as the baud rate clock source in mode 2, the baud rate is determined by the SMOD value according to the following equations.

$$B = F_{osc} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{16} \quad (SMOD = 0)$$

$$B = F_{osc} \times \frac{1}{2} \times \frac{1}{16} \quad (SMOD = 1)$$

where B is the baud rate and  $F_{osc}$  is the fundamental oscillator ( $XTAL1 \cdot 2$ ) frequency.

##### 4.6.3.3.3 Mode 2 transmit operation

The basic clock, TXCLOCK in Figure 4-27, for transmitting purposes is obtained from a hexadecimal counter overflow which employs the frequency of  $1/2XTAL1 \cdot 2$  divided by 2 when  $SMOD = 0$  or the unchanged  $1/2XTAL1 \cdot 2$  frequency when  $SMOD = 1$  as the clock.

Transmission is started when transmission data is written in SBUF. The start bit, the eight SBUF data bits (with the LSB first), TB8, and the stop bit are thus transmitted sequentially from the TXD synchronized with the basic clock.

As soon as the TB8 output has been completed, the transmit circuit is initialized, and the TI flag is set at the first M1 · S3 cycle after the completion of that output.

##### 4.6.3.3.4 Mode 2 receive operation

The receive circuit timing is generated by a hexadecimal counter which employs the clock frequency of  $1/2XTAL1 \cdot 2$  divided by 2 when  $SMOD = 0$  or the unchanged  $1/2XTAL1 \cdot 2$  frequency when  $SMOD = 1$  as the clock. The input data received from the RXD is bit synchronized. That is, at the same time that reception is started following input of the start bit, the hexadecimal counter commences to count up, and with one complete round of the hexadecimal counter corresponding to one bit of received data, reception is continued by the receive circuit. Therefore, the reception data baud rate must be equal to the period of a single round of the hexadecimal counter.

The change in RXD from level "1" to "0" is regarded as the trigger to commence reception.

When this "1" to "0" change in RXD is detected, the hexadecimal counter which had been in reset mode commences to count up. When the hexadecimal counter is in state 7, 8, and 9, the start bit is sampled, and is accepted as valid if at least two of the three sampled values are "0", thereby enabling data reception to continue. If two or three of the sampled values are "1", the start bit becomes invalid, and the receive circuit is initialized when the hexadecimal counter reaches state 10.

The reception data is sampled when the hexadecimal counter is in state 7, 8, and 9. The most common value of the three sampled values is read sequentially as data into the input shift register.

If the conditions stated below are satisfied when the hexadecimal counter is in state 10 during the period of the multi-purpose data bit, the input shift register data (the LSB being read first) is loaded into SBUF and the sampled multi-purpose data bit is read into RB8.

And when the hexadecimal counter is in state 10 during the period of the stop bit, the receive circuit is initialized.

Conditions: (1) RI = "0"

(2) SM2 = "0", or

SM2 = "1" and sampled multi-purpose data bit = "1"

The RI flag is set at the first M1 · S3 cycle after the initialization.

If the above conditions are not satisfied when the hexadecimal counter is in state 10 during the multi-purpose data bit interval, the received data is disregarded, the SBUF, RB8, and RI flags remain unchanged, and the receive circuit is initialized when the hexadecimal counter is in state 10 during the stop bit interval.

Since the receive circuit is divided into two stages, input shift register and SBUF, processing of the previous receive data may be completed within the interval up to the multi-purpose data bit period of the next frame.



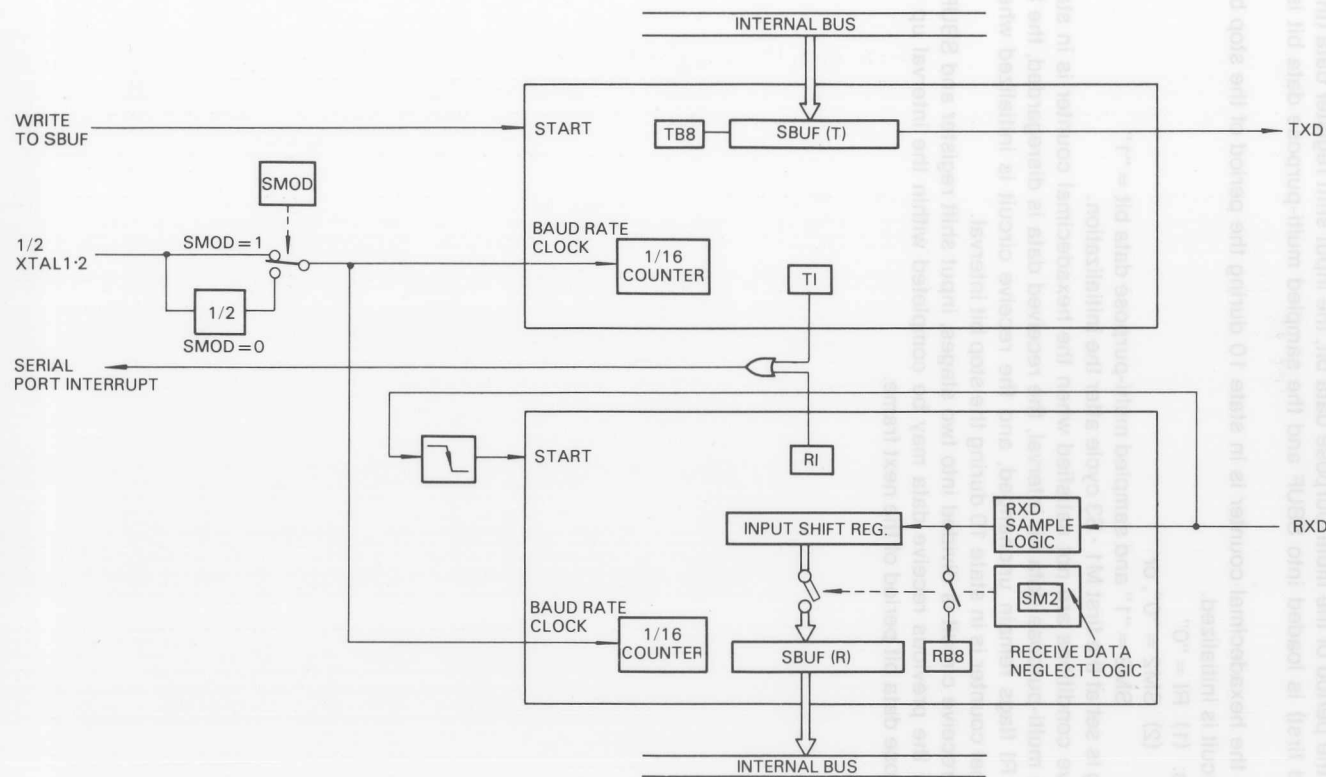


Figure 4-26 Serial port (mode 2)

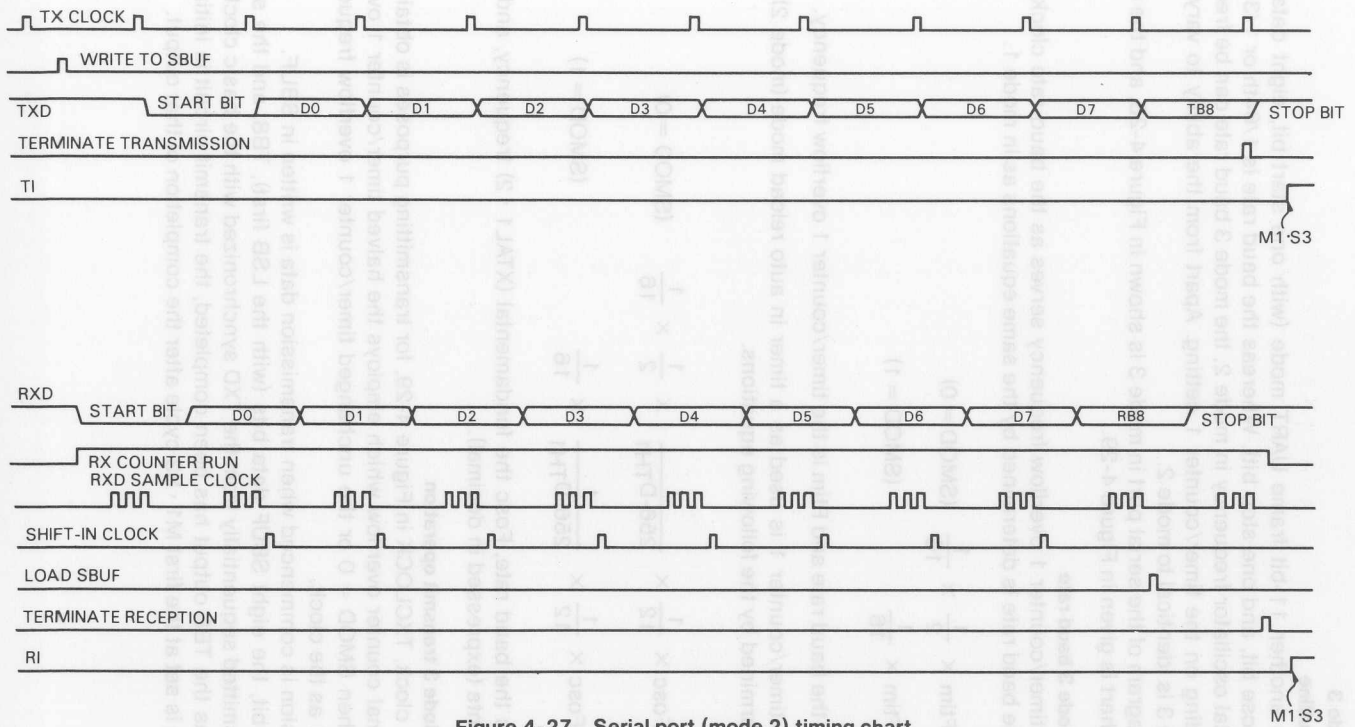


Figure 4-27 Serial port (mode 2) timing chart

**4.6.3.4 Mode 3****4.6.3.4.1 Outline**

Mode 3 is another 11-bit frame UART mode (with one start bit, eight data bits, one multi-purpose bit, and one stop bit). Whereas the baud rate is 1/64th or 1/32nd of the fundamental oscillator frequency in mode 2, the mode 3 baud rate can be freely selected depending on the timer/counter 1 setting. Apart from the ability to vary the baud rate, mode 3 is identical to mode 2.

A block diagram of the serial port in mode 3 is shown in Figure 4-28, and the operational timing chart is given in Figure 4-29.

**4.6.3.4.2 Mode 3 baud rate**

Since the timer/counter 1 overflow frequency serves as the baud rate clock source in mode 3, the baud rate is determined by the same equations as in mode 1.

$$B = F_{tim} \times \frac{1}{2} \times \frac{1}{16} \quad (SMOD = 0)$$

$$B = F_{tim} \times \frac{1}{16} \quad (SMOD = 1)$$

where B is the baud rate and  $F_{tim}$  is the timer/counter 1 overflow frequency.

And when timer/counter 1 is used as a timer in auto reload mode (mode 2), the baud rate is determined by the following equations.

$$B = F_{osc} \times \frac{1}{12} \times \frac{1}{256-D_{TH1}} \times \frac{1}{2} \times \frac{1}{16} \quad (SMOD = 0)$$

$$B = F_{osc} \times \frac{1}{12} \times \frac{1}{256-D_{TH1}} \times \frac{1}{16} \quad (SMOD = 1)$$

where B is the baud rate,  $F_{osc}$  the fundamental (XTAL1 · 2) frequency, and  $D_{TH1}$  the TH1 contents (expressed in decimal).

**4.6.3.4.3 Mode 3 transmit operation**

The basic clock, TXCLOCK in Figure 4-29, for transmitting purposes is obtained from a hexadecimal counter overflow which employs the halved timer/counter 1 overflow frequency when  $SMOD = 0$  or the unchanged timer/counter 1 overflow frequency when  $SMOD = 1$  as the clock.

Transmission is commenced when transmission data is written in SBUF.

The start bit, the eight SBUF data bits (with the LSB first), TB8, and the stop bit are thus transmitted sequentially from the TXD synchronized with the basic clock.

As soon as the TB8 output has been completed, the transmit circuit is initialized, and the TI flag is set at the first M1 · S3 cycle after the completion of that output.

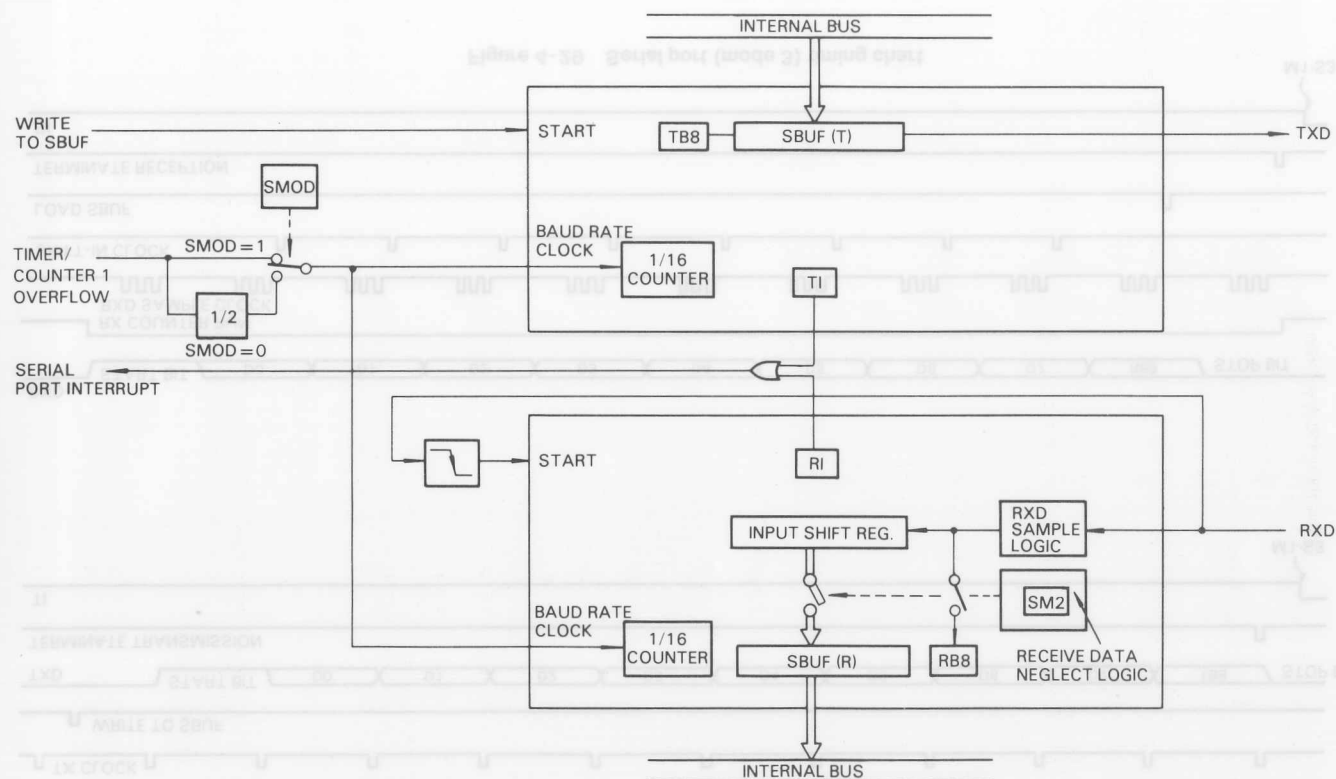


Figure 4-28 Serial port (mode 3)

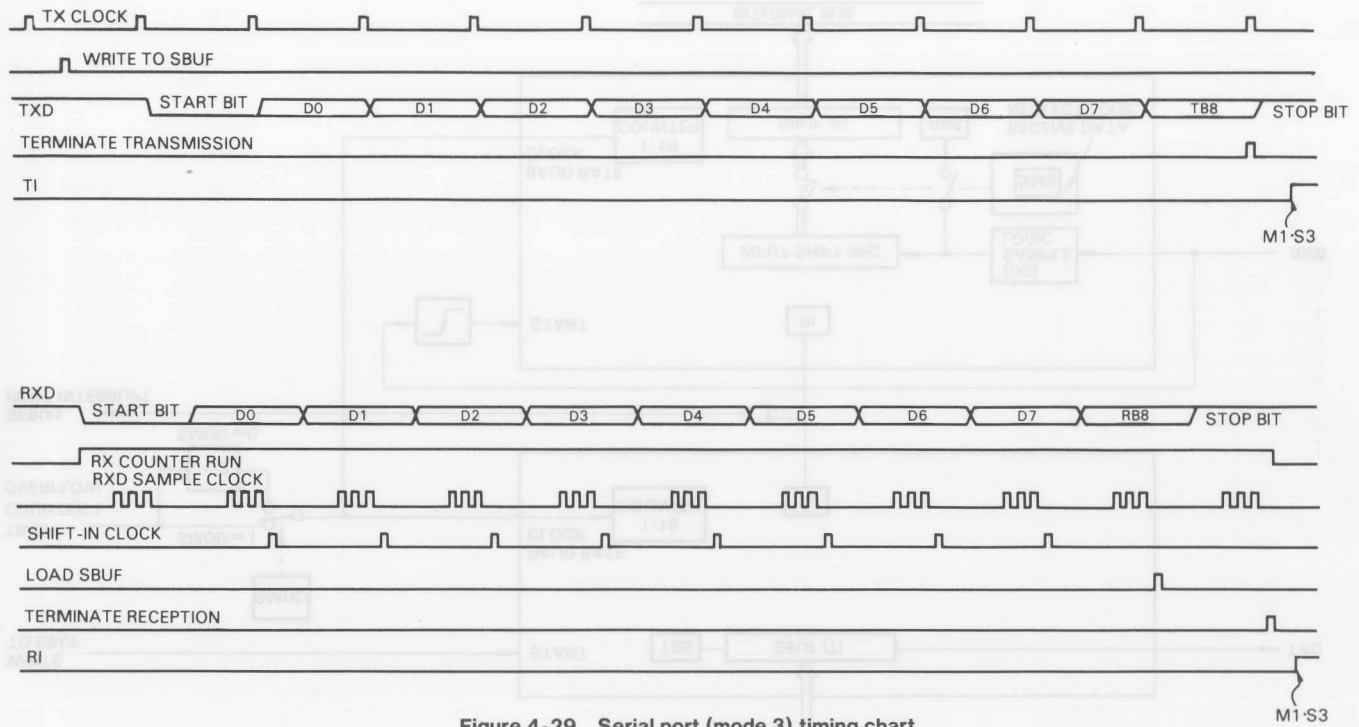


Figure 4-29 Serial port (mode 3) timing chart

**4.6.3.4.4 Mode 3 receive operation**

The receive circuit timing is generated by a hexadecimal counter which employs the halved timer/counter 1 overflow frequency when SMOD = 0 or the unchanged timer/counter 1 overflow frequency when SMOD = 1 as the clock. The input data received from the RXD is bit synchronized. That is, at the same time that reception is started following input of the start bit, the hexadecimal counter commences to count up, and with one complete round of the hexadecimal counter corresponding to one bit of received data, reception is continued by the receive circuit. Therefore, timer/counter 1 must be set so that the period of a single round of the hexadecimal counter is equal to the reception data baud rate.

The RXD change from "1" to "0" is regarded as the beginning of the start bit for reception.

When this "1" to "0" change in RXD is detected, the hexadecimal counter which had been in reset mode commences to count up. When the hexadecimal counter is in state 7, 8, and 9, the start bit is sampled, and is accepted as valid if at least two of the three sampled values are "0", thereby enabling data reception to continue. If two or three of the sampled values are "1", the start bit becomes invalid, and the receive circuit is re-initialized when the hexadecimal counter reaches state 10.

The reception data is sampled when the hexadecimal counter is in state 7, 8, and 9, and the more common value of the three sampled values is read sequentially as data into the input shift register.

If the conditions stated below are satisfied when the hexadecimal counter is in state 10 during the period of the multi-purpose data bit, the input shift register data (the LSB being read first) is loaded into SBUF, and the sampled multi-purpose data bit is read into RB8.

And when the hexadecimal counter is in state 10 during the period of the stop bit, the receive circuit is initialized.

Conditions: (1) RI = "0"

(2) SM2 = "0", or

SM2 = "1" and sampled multi-purpose data bit = "1"

The RI flag is set at the first M1 · S3 cycle after that.

If the above conditions are not satisfied when the hexadecimal counter is in state 10 during the multi-purpose data bit interval, the received data is disregarded, the SBUF, RB8, and RI flags remain unchanged, and the receive circuit is initialized when the hexadecimal counter is in state 10 during the stop bit interval.

Since the receive circuit is divided into two stages (input shift register and SBUF), processing of the previous receive data may be completed within the interval up to the multi-purpose data bit period of the next frame.

#### 4.6.4 Serial port application examples

##### 4.6.4.1 I/O extension

I/O extension can be achieved by using the serial port in mode 0. An input extension example is shown in Figure 4-30 and the corresponding timing chart is shown in Figure 4-31. The parallel input is latched into 74LS165 with the output of the latch pulse PX.X. Then, this is followed by REN = 1 and RI = 0 setting, and shift in of 74LS165 data.

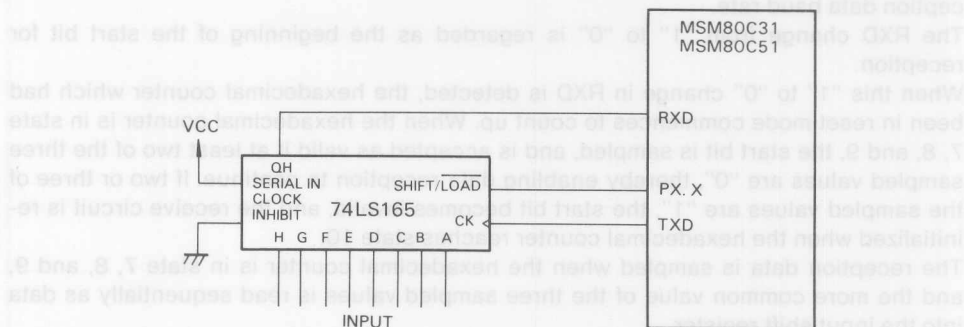


Figure 4-30 Input extension example

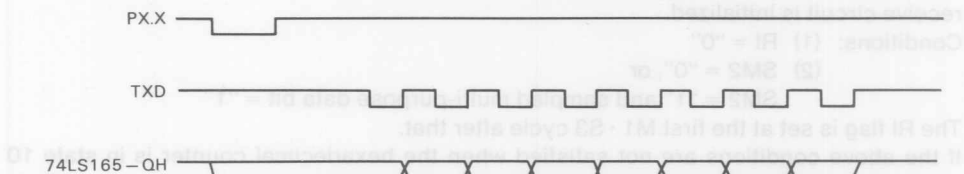


Figure 4-31 Input extension example timing chart

An output extension example is shown in Figure 4-32 and the corresponding timing chart is shown in Figure 4-33. After output data has been written into SBUF and the output sequence completed, the latch pulse output from PX.X is obtained and the 74LS164 data is shifted to 74LS373.

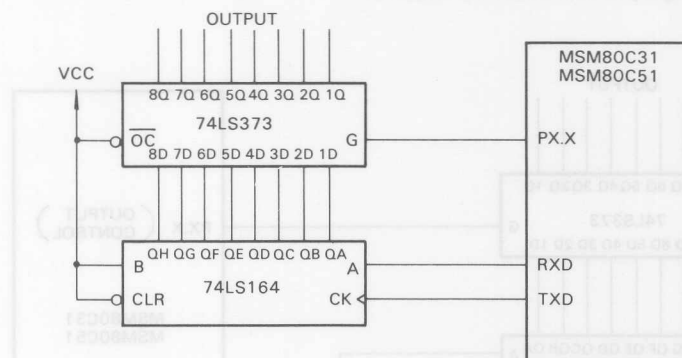


Figure 4-32 Output extension example

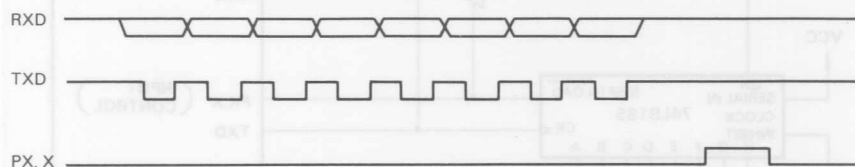
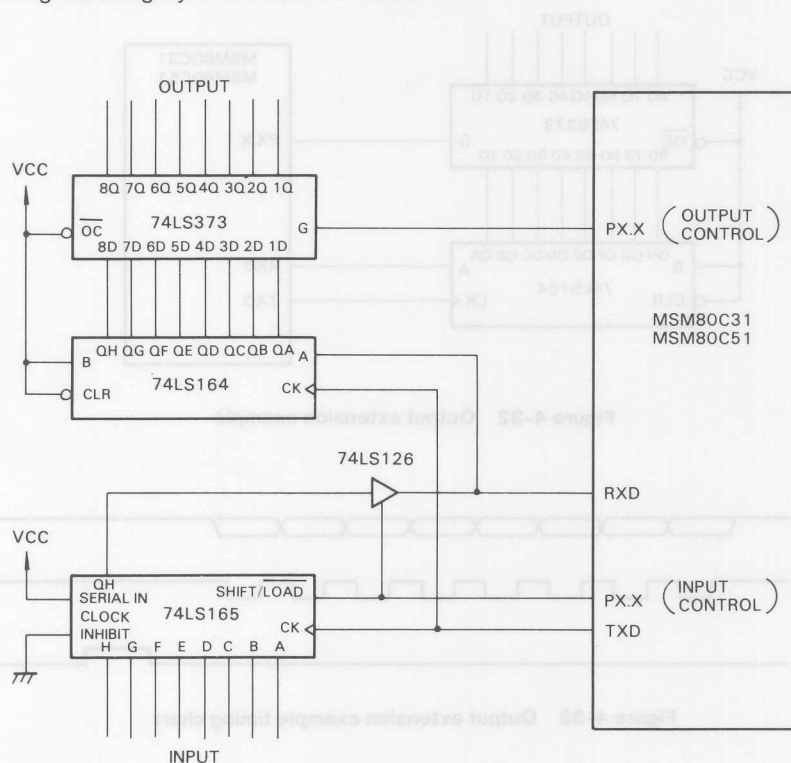


Figure 4-33 Output extension example timing chart

An input/output extension example is shown in Figure 4-34 and the corresponding timing chart is shown in Figure 4-35. When input data is applied, INPUT CONTROL is changed from "0" to "1", and the parallel input is latched. This is then followed by REN = 1 and RI = 0 settings, and shift in of 74LS165 data. INPUT CONTROL is returned to "0" after the input has been completed. Since INPUT CONTROL is connected to the 74LS126 control pin, the MSM80C31/MSM80C51 switches the 74LS126 output to high impedance when 74LS165 input data is not being applied, thereby preventing collision between the 74LS126 and MSM80C31/MSM80C51 outputs.



When output data is generated, after the output data is written into 74LS164, an output latch pulse is generated from OUTPUT CONTROL, and the 74LS164 data is transferred to 74LS373. Although the 74LS164 data is changed to parallel input data when 74LS165 data is passed to MSM80C31/MSM80C51, an output latch pulse is generated only when output data is obtained from MSM80C31/MSM80C51, thereby preserving the integrity of the data in 74LS373.



**Figure 4-34** Input/output extension example

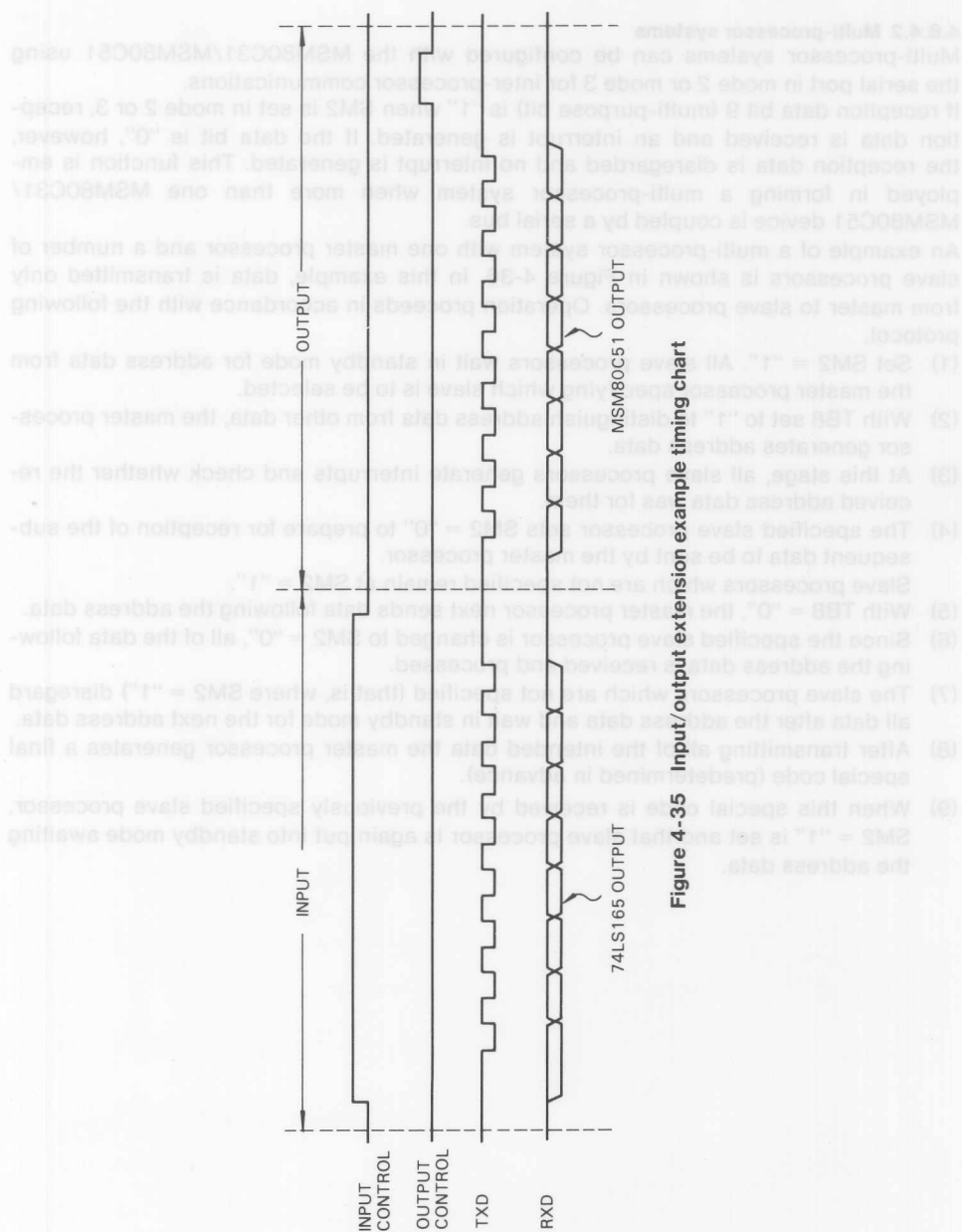


Figure 4-35 Input/output extension example timing chart

In all examples, additional multiple bit I/O extension is made possible by multiple cascade connections of 74LS164 or 74LS165.

#### 4.6.4.2 Multi-processor systems

Multi-processor systems can be configured with the MSM80C31/MSM80C51 using the serial port in mode 2 or mode 3 for inter-processor communications.

If reception data bit 9 (multi-purpose bit) is "1" when SM2 is set in mode 2 or 3, reception data is received and an interrupt is generated. If the data bit is "0", however, the reception data is disregarded and no interrupt is generated. This function is employed in forming a multi-processor system when more than one MSM80C31/MSM80C51 device is coupled by a serial bus.

An example of a multi-processor system with one master processor and a number of slave processors is shown in Figure 4-36. In this example, data is transmitted only from master to slave processors. Operation proceeds in accordance with the following protocol.

- (1) Set SM2 = "1". All slave processors wait in standby mode for address data from the master processor specifying which slave is to be selected.
- (2) With TB8 set to "1" to distinguish address data from other data, the master processor generates address data.
- (3) At this stage, all slave processors generate interrupts and check whether the received address data was for them.
- (4) The specified slave processor sets SM2 = "0" to prepare for reception of the subsequent data to be sent by the master processor.  
Slave processors which are not specified remain at SM2 = "1".
- (5) With TB8 = "0", the master processor next sends data following the address data.
- (6) Since the specified slave processor is changed to SM2 = "0", all of the data following the address data is received and processed.
- (7) The slave processors which are not specified (that is, where SM2 = "1") disregard all data after the address data and wait in standby mode for the next address data.
- (8) After transmitting all of the intended data the master processor generates a final special code (predetermined in advance).
- (9) When this special code is received by the previously specified slave processor, SM2 = "1" is set and that slave processor is again put into standby mode awaiting the address data.

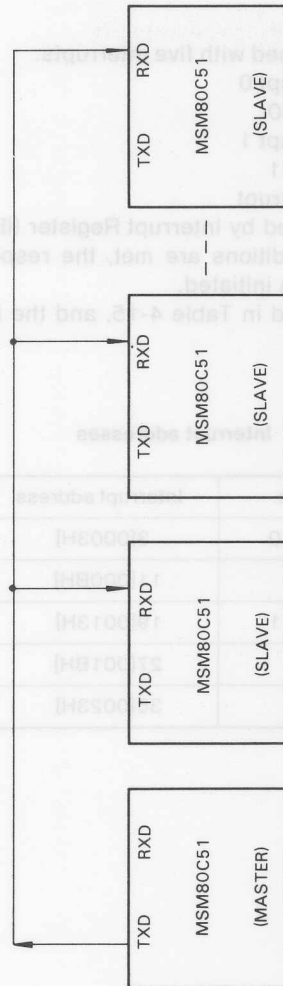


Figure 4-36 Multi-processor system example

## 4.7 Interrupts

### 4.7.1 Outline

MSM80C31/MSM80C51 is equipped with five interrupts.

1.  $\overline{\text{INT0}}$  : External interrupt 0
2.  $\text{TM0}$  : Timer interrupt 0
3.  $\overline{\text{INT1}}$  : External interrupt 1
4.  $\text{TM1}$  : Timer interrupt 1
5.  $\text{SI/O}$  : Serial port interrupt

These five interrupts are controlled by Interrupt Register (IE) and Priority Register (IP). When the relevant interrupt conditions are met, the respective interrupt address is called and the interrupt process is initiated.

The interrupt addresses are listed in Table 4-15, and the interrupt control equivalent circuit is shown in Figure 4-37.

**Table 4-15 Interrupt addresses**

	Interrupt source	Interrupt address
1	External interrupt 0	3[0003H]
2	Timer interrupt 0	11[000BH]
3	External interrupt 1	19[0013H]
4	Timer interrupt 1	27[001BH]
5	Serial port	35[0023H]



**4.7.2 Interrupt enable register (IE)**

The function of the interrupt enable register (IE, A8H) is to enable or disable interrupt processing whenever an interrupt request is made.

To execute the intended interrupt processing, the interrupt is enabled by setting "1" in the corresponding bit in the interrupt enable register, and on requested, the interrupt is processed.

Requested interrupts are disabled if the corresponding bit is "0", and the interrupt processing is not executed.

The contents of the interrupt enable register (IE) are shown in Table 4-16.

**Table 4-16 Interrupt enable register (IE, A8H)**

Bit	7	6	5	4	3	2	1	0
Flag	EA	—	—	ES	ET1	EX1	ET0	EX0

- EX0 : External interrupt 0 control bit  
Interrupt enabled when "1", disabled when "0".
- ET0 : Timer interrupt 0 control bit  
Interrupt enabled when "1", disabled when "0".
- EX1 : External interrupt 1 control bit  
Interrupt enabled when "1", disabled when "0".
- ET1 : Timer interrupt 1 control bit  
Interrupt enabled when "1", disabled when "0".
- ES : Serial port interrupt control bit  
Interrupt enabled when "1", disabled when "0".
- : Reserve bit for output of "1" when read.
- : Reserve bit for output of "1" when read.
- EA : Interrupt control bit for all five interrupts (EX0, ET0, EX1, ET1, and ES)  
When EA is "1", interrupt processing is commenced if interrupt conditions are met for any one of the five interrupts.  
When EA is "0", interrupt processing is not commenced even if interrupt conditions are met for any one of the five interrupts.

#### 4.7.3 Interrupt priority register (IP)

The function of the interrupt priority register (IP, B8H) is to allocate rights to commence processing of interrupts on a priority basis when an interrupt is requested.

Each interrupt source can be programmed to higher priority level by setting the bit corresponding to the interrupt source in the interrupt priority register (IP). When the interrupt conditions are satisfied for an interrupt with higher priority, even if another interrupt with lower priority (priority bit being "0") is already being processed, that processing is suspended, and the higher priority interrupt is serviced first. Note that once processing of a higher priority interrupt has been commenced, processing of the next interrupt cannot start until the current interrupt is completed.

The contents of the interrupt priority register are given in Table 4-17.

**Table 4-17 Interrupt priority register (IP, B8H)**

Bit	7	6	5	4	3	2	1	0
Flag	—	—	—	PS	PT1	PX1	PT0	PX0

- PX0 : External interrupt 0 priority bit
- PT0 : Timer interrupt 0 priority bit
- PX1 : External interrupt 1 priority bit
- PT1 : Timer interrupt 1 priority bit
- PS : Serial port interrupt priority bit
- : Reserve bit for output of "1" when read.
- : Reserve bit for output of "1" when read.
- : Reserve bit for output of "1" when read.

4



#### 4.7.3.1 Priority interrupt process flow

The flow of interrupt process when a priority interrupt is generated after processing has commenced on a non-priority interrupt generated during execution of a main routine program is outlined in Figure 4-38.

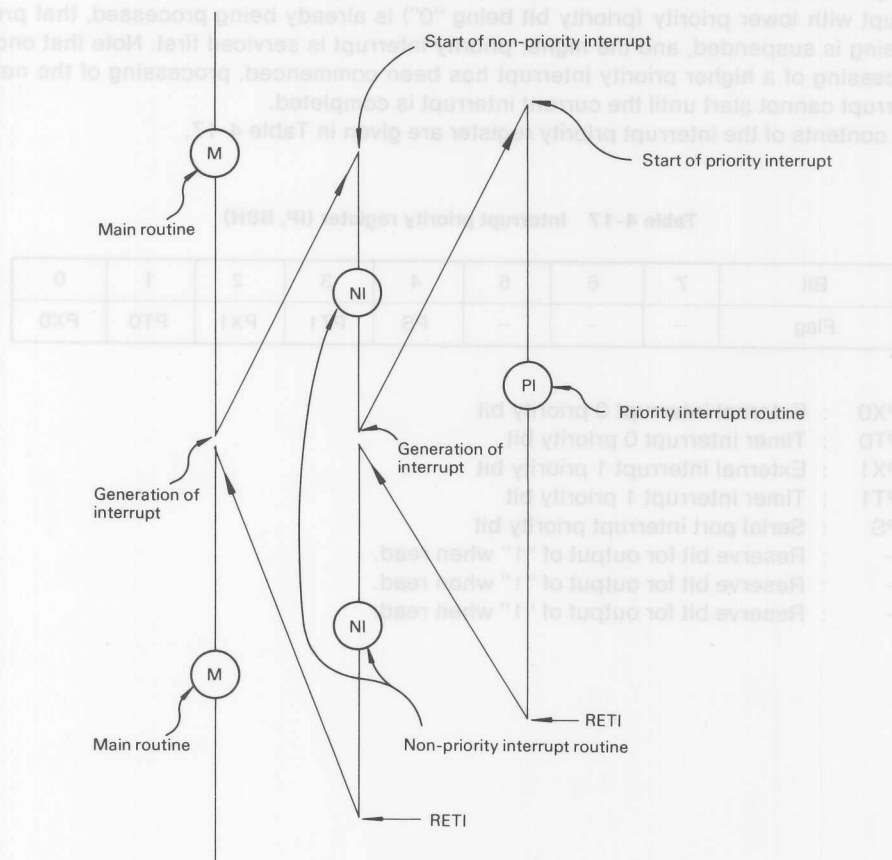


Figure 4-38 Interrupt processing flow chart during priority circuit operation

**4.7.3.2 Interrupt priority when priority register (IP) contents are all "0"**

The interrupt priority when the priority register (IP, B8H) contents are all "0" indicates the priority in which a certain interrupt is processed in preference to other interrupts when interrupt requests are generated simultaneously. As can be seen from Table 4-18, the external interrupt 0 has the highest priority while the serial port interrupt has the lowest priority.

If all priority bits are "0" and a certain interrupt is in process, it will not be preempted by another interrupt regardless of its priority.

The same operational preferences as described above also exist when all priority bits are "1".

INT0		INT1		Timer																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
------	--	------	--	-------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

#### 4.7.4 Detection of external interrupt signals $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$

##### 4.7.4.1 Outline of $\overline{\text{INT}}$ signal detection

The external interrupt signals 0 and 1 can be set to level-detect or trigger-detect mode by the ITO and IT1 data values of bits 0 and 2 in the timer control register (TCON 88H) as indicated in Table 4-19.

Table 4-19 TCON [88H] register

	Timer				INT1		INT0	
Bit	7	6	5	4	3	2	1	0
Flag	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Set						●		●

##### 4.7.4.2 External interrupt signal 0 and 1 level detection

When the bit-0, IT0, in the timer control register (TCON 88H) is "0", external interrupt 0 is detected on the basis of the level of the signal. And when the bit 2, IT1, is "0", external interrupt 1 is also detected on the basis of the signal's level.

In the level detection mode, external interrupts 0 and 1 are detected by the equivalent circuit shown in Figure 4-39. When the level of the external interrupt pin is "0" at S5 timing, the level is latched and the Q output becomes "1". The latched external interrupt signal sets the external interrupt flag in the timer control register (TCON) at the S3 timing. This interrupt flag set by external interrupt signal is always reset at S6 in the last machine cycle of instruction execution, effectively being equivalent to a "level sense" operation. The cycle width of the respective "0" and "1" levels of the external interrupt signal applied to the external interrupt pin in this case must be at least 12 times (12T) XTAL1/2 oscillator clock cycle time T.

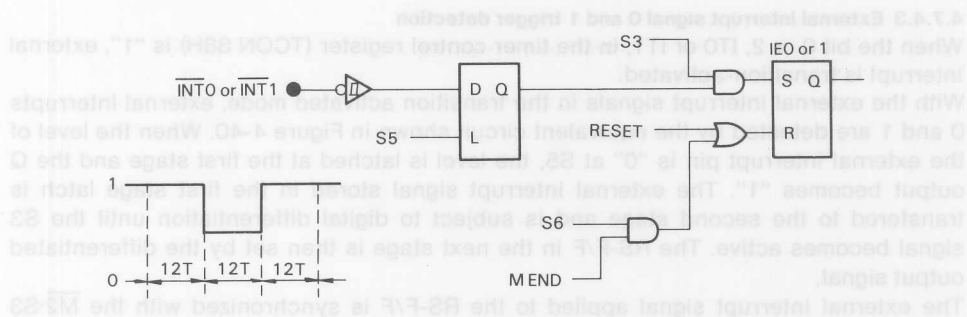


Figure 4-39 Interrupt level input equivalent circuit for IT bit "0"

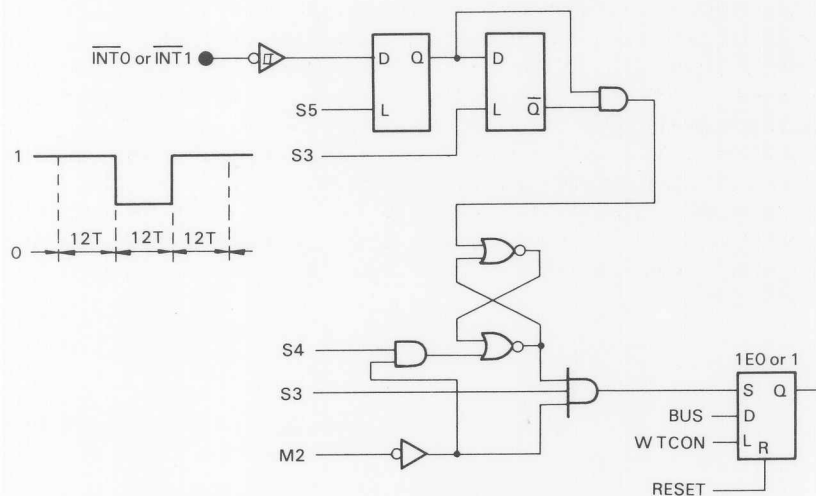


Figure 4-40 Interrupt edge input equivalent circuit for IT bit "1"

#### 4.7.4.3 External interrupt signal 0 and 1 trigger detection

When the bit 0 or 2, IT0 or IT1, in the timer control register (TCON 88H) is "1", external interrupt is transition-activated.

With the external interrupt signals in the transition activated mode, external interrupts 0 and 1 are detected by the equivalent circuit shown in Figure 4-40. When the level of the external interrupt pin is "0" at S5, the level is latched at the first stage and the Q output becomes "1". The external interrupt signal stored in the first stage latch is transferred to the second stage and is subject to digital differentiation until the S3 signal becomes active. The RS-F/F in the next stage is then set by the differentiated output signal.

The external interrupt signal applied to the RS-F/F is synchronized with the  $\overline{M2}\cdot S3$  signal to be applied as a trigger for the external interrupt flag in the timer control register (TCON). The RS-F/F is subsequently reset at  $\overline{M2}\cdot S4$  and it waits for the next interrupt. Note that the next interrupt signal cannot be detected until the first stage latch detects a level "0" to level "1" transition on the external interrupt signal.

The cycle width of the respective "0" and "1" levels of the external interrupt signal applied to the external interrupt pin in this case must be at least 12 times (12T) XTAL1/2 oscillator clock cycle time T.

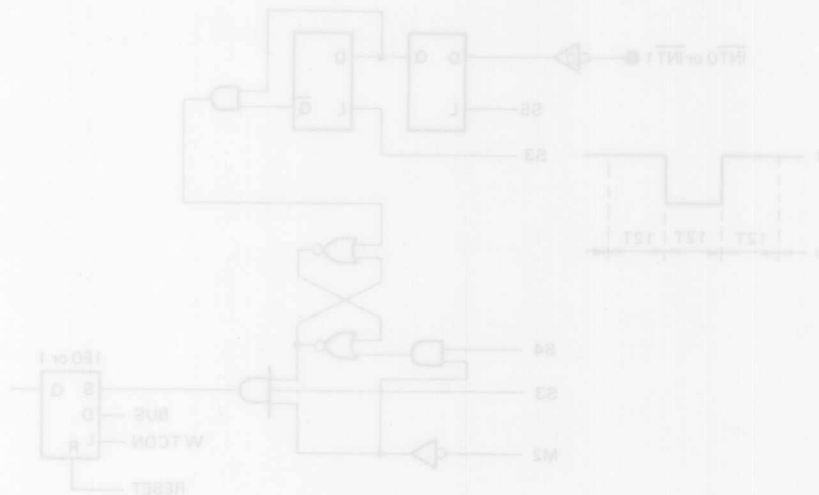


Figure 4-40 Internal edge input equivalent circuit for IT bit "1"

#### 4.7.5 MSM80C31/MSM80C51 interrupt response time charts

##### 4.7.5.1 Interrupt response time chart when interrupt conditions are satisfied during execution of ordinary instruction

If interrupt conditions are satisfied during execution of an ordinary instruction (which do not manipulate IE or IP) in the main routine, the MSM80C31/MSM80C51 calls the interrupt address in the next cycle following completion of the ordinary instruction. The time chart is given in Figure 4-41.

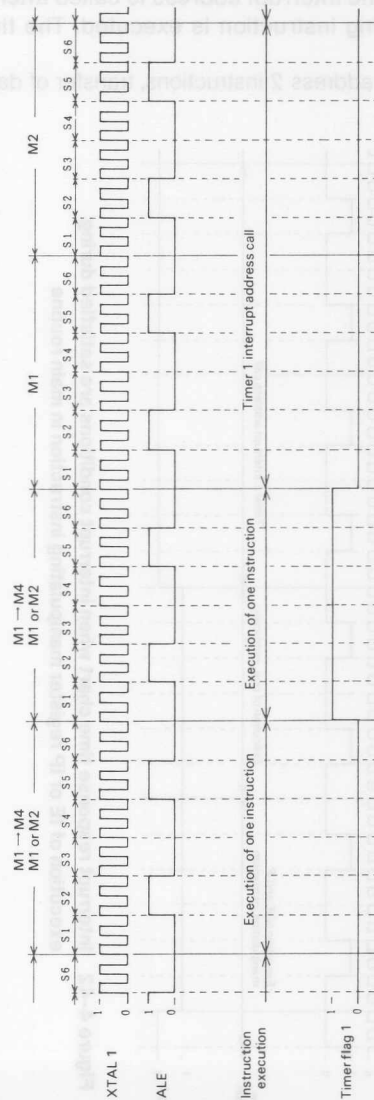


Figure 4-41 Interrupt response time chart when interrupt conditions are satisfied during execution of ordinary instruction in main routine

#### 4.7.5.2 Interrupt response time chart when interrupt conditions are satisfied during execution of IE or IP register operation instruction

If interrupt conditions are satisfied during execution of an instruction used to manipulate the interrupt enable register (IE) or the interrupt priority register (IP) in the main routine, the MSM80C31/MSM80C51 reactivates the interrupt mask circuit in the next cycle following completion of the register manipulation instruction. If interrupt conditions are met as a result of the re-interrupt mask, the interrupt address is called in the next cycle. That is, if the interrupt conditions are satisfied during execution of the IE or the IP manipulating instruction, the interrupt address is called after the next instruction following the register manipulating instruction is executed. The time chart is given in Figure 4-42.

\* In the MOV data address 1, data address 2 instructions, transfer of data to another register from IE or IP is an exception.

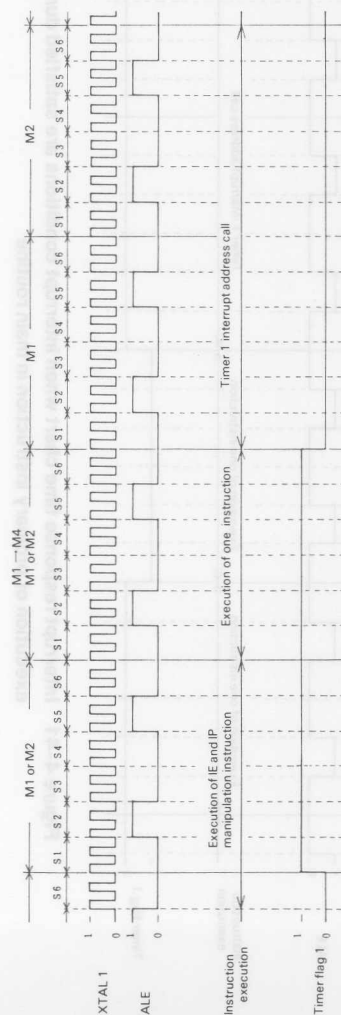


Figure 4-42 Interrupt response time chart when interrupt conditions are satisfied during execution of IE or IP register manipulating instruction in main routine

#### 4.7.5.3 Interrupt response time chart when an ordinary instruction is executed after temporarily returning to the main routine from continuous interrupt processing

If an ordinary instruction (that is, not an IE or IP manipulating instruction) is executed after returning to the main routine following execution of the interrupt processing end instruction RETI, and if the next interrupt conditions were met during execution of an interrupt processing routine, the MSM80C31/MSM80C51 calls the interrupt address in the next cycle following execution of one main routine instruction. The same occurs when interrupt conditions are satisfied during execution of the first main routine instruction after returning to that routine from the interrupt processing routine. The time chart is shown in Figure 4-43.

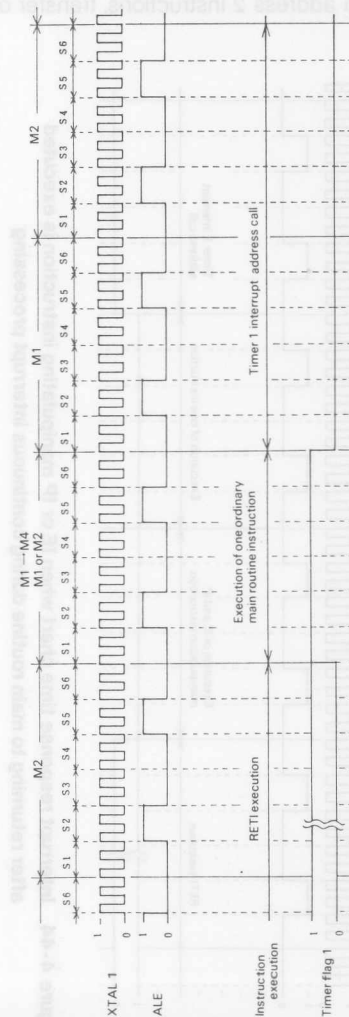


Figure 4-43 Interrupt response time chart when ordinary instruction is executed after returning to main routine during continuous interrupt processing



#### 4.7.5.4 Interrupt response time chart when an IE or IP manipulating instruction is executed after temporarily returning to the main routine from continuous interrupt processing

If the "next" interrupt conditions are satisfied during execution of an interrupt processing routine, and if the interrupt terminating instruction RETI is then executed followed by a return to the main routine where an interrupt enable register (IE) or interrupt priority register (IP) manipulating instruction is executed, the MSM80C31/MSM80C51 activates the interrupt mask circuit in the next cycle following execution of the register manipulating instruction. And if interrupt conditions are met as a result of the re-interrupt mask, the interrupt address is called in the next cycle. That is, if the instruction executed in the main routine manipulates either IE or IP, the interrupt address is called after two instructions are executed. The time chart is shown in Figure 4-44.

\* In the MOV data address 1, data address 2 instructions, transfer of data to another register from IE or IP is an exception.

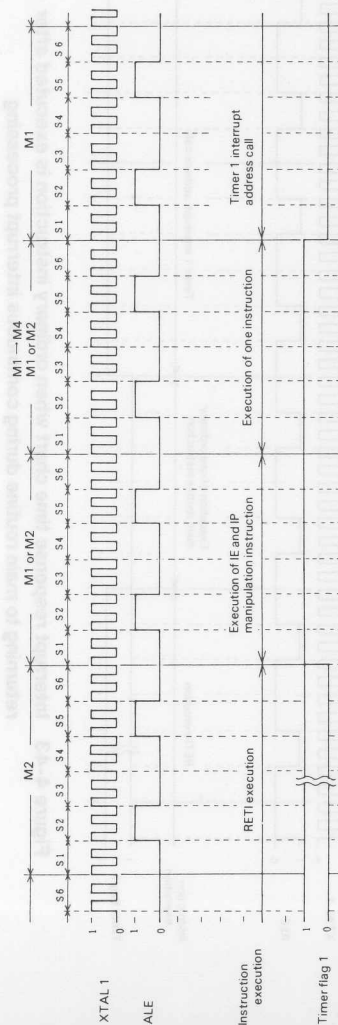


Figure 4-44 Interrupt response time chart when IE or IP manipulating instruction is executed after returning to main routine during continuous interrupt processing

## 4.8 CPU "Power Down"

### 4.8.1 Outline

The CPU power down function operates in two modes – idle mode and power down mode. In the idle mode (IDLE) where "1" is set in bit 0 (IDL) of the power control register (PCON) by software, XTAL1·2 operation is continued but the clock to the CPU control section is stopped, thereby halting CPU operations. In the power down mode (PD) where "1" is set in bit 1 (PD) of the power control register (PCON) by software, both the XTAL1·2 operation and the CPU operation is stopped.

The device can be forced out of the power down mode by either a reset or an interrupt.

### 4.8.2 Idle mode (IDLE) setting

Idle mode is set when "1" is set in bit 0 (IDL) of the power control register (PCON 87H). The circuit connection involved in this setting is shown in Figure 4-45.

In the idle mode, the clock to the CPU control section is stopped and the CPU operations are halted. But since XTAL1·2 operations are maintained, the serial port, interrupt circuits, and timer/counters 0 and 1 remain operative. The CPU pin status during the idle mode is outlined in Table 4-20, and the corresponding time charts for initiating the idle mode are shown in Figures 4-46 and 4-47.

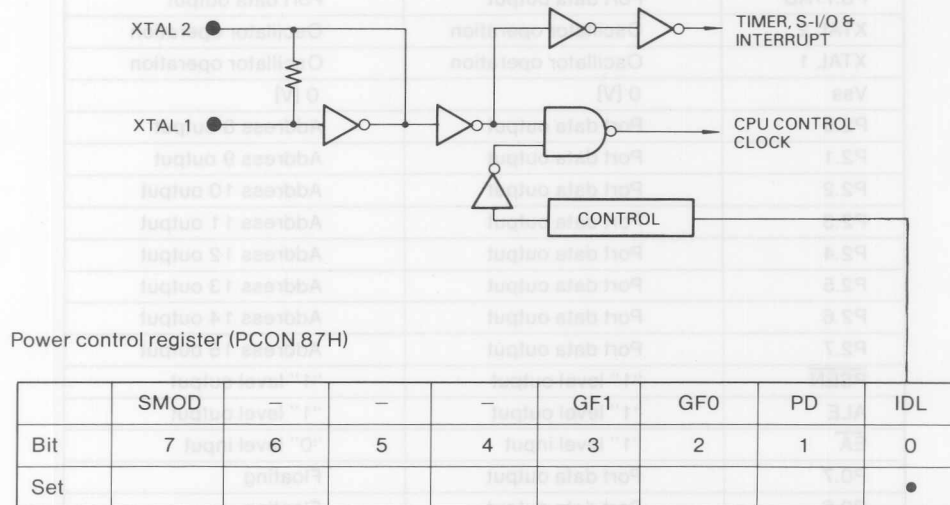


Figure 4-45 Idle mode equivalent circuit

Table 4-20 CPU pin status in idle mode

Name	Internal ROM	External ROM
P1.0	Port data output	Port data output
P1.1	Port data output	Port data output
P1.2	Port data output	Port data output
P1.3	Port data output	Port data output
P1.4	Port data output	Port data output
P1.5	Port data output	Port data output
P1.6	Port data output	Port data output
P1.7	Port data output	Port data output
RESET	"0" level input	"0" level input
P3.0/RXD	Port data output	Port data output
P3.1/TXD	Port data output	Port data output
P3.2/ $\overline{\text{INT0}}$	Port data output	Port data output
P3.3/ $\overline{\text{INT1}}$	Port data output	Port data output
P3.4/T0	Port data output	Port data output
P3.5/T1	Port data output	Port data output
P3.6/ $\overline{\text{WR}}$	Port data output	Port data output
P3.7/ $\overline{\text{RD}}$	Port data output	Port data output
XTAL 2	Oscillator operation	Oscillator operation
XTAL 1	Oscillator operation	Oscillator operation
Vss	0 [V]	0 [V]
P2.0	Port data output	Address 8 output
P2.1	Port data output	Address 9 output
P2.2	Port data output	Address 10 output
P2.3	Port data output	Address 11 output
P2.4	Port data output	Address 12 output
P2.5	Port data output	Address 13 output
P2.6	Port data output	Address 14 output
P2.7	Port data output	Address 15 output
$\overline{\text{PSEN}}$	"1" level output	"1" level output
ALE	"1" level output	"1" level output
$\overline{\text{EA}}$	"1" level input	"0" level input
P0.7	Port data output	Floating
P0.6	Port data output	Floating
P0.5	Port data output	Floating
P0.4	Port data output	Floating
P0.3	Port data output	Floating
P0.2	Port data output	Floating
P0.1	Port data output	Floating
P0.0	Port data output	Floating
Vcc	+2.5 ~ +6 [V]	+2.5 ~ +6 [V]

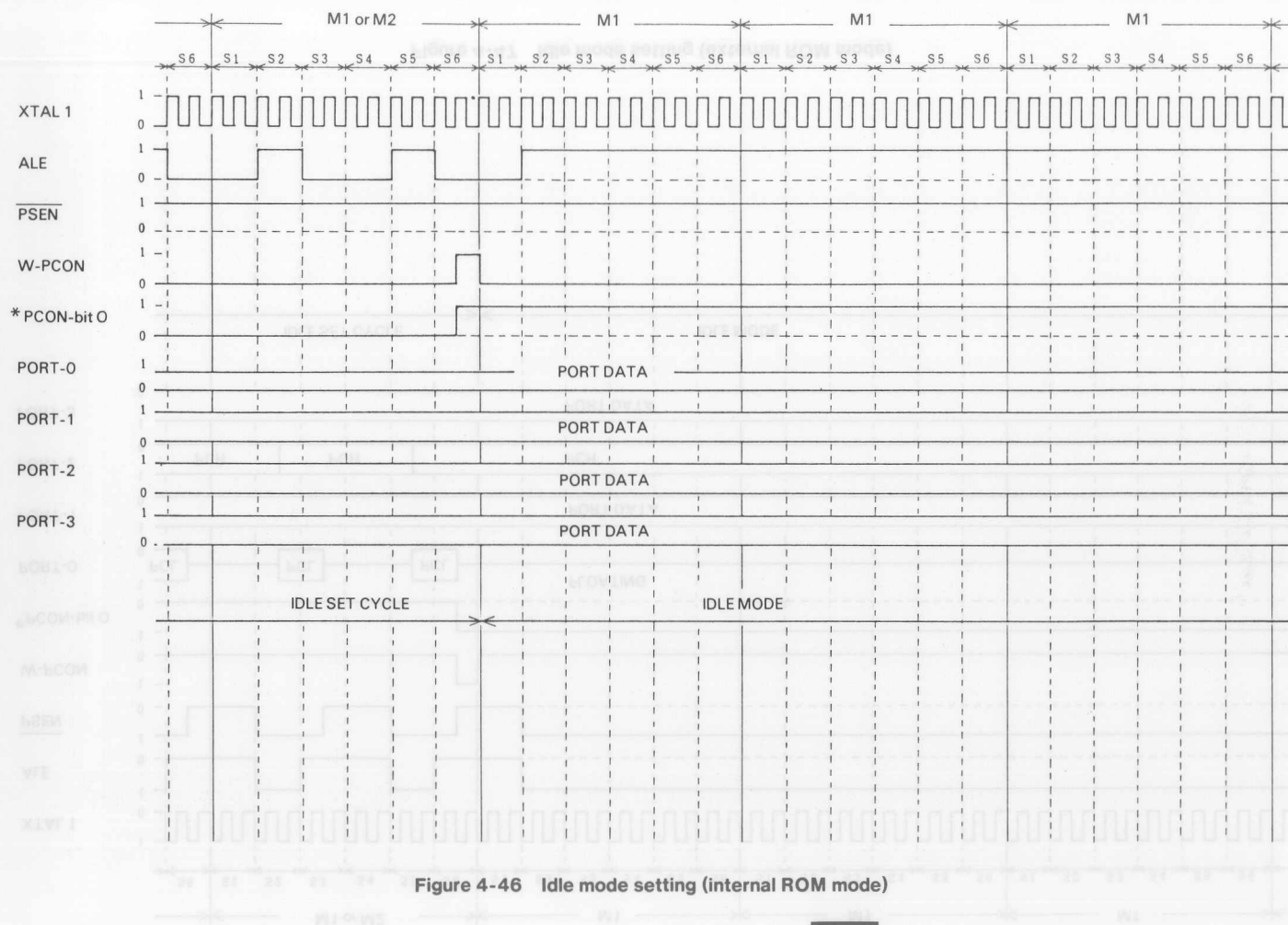


Figure 4-46 Idle mode setting (internal ROM mode)

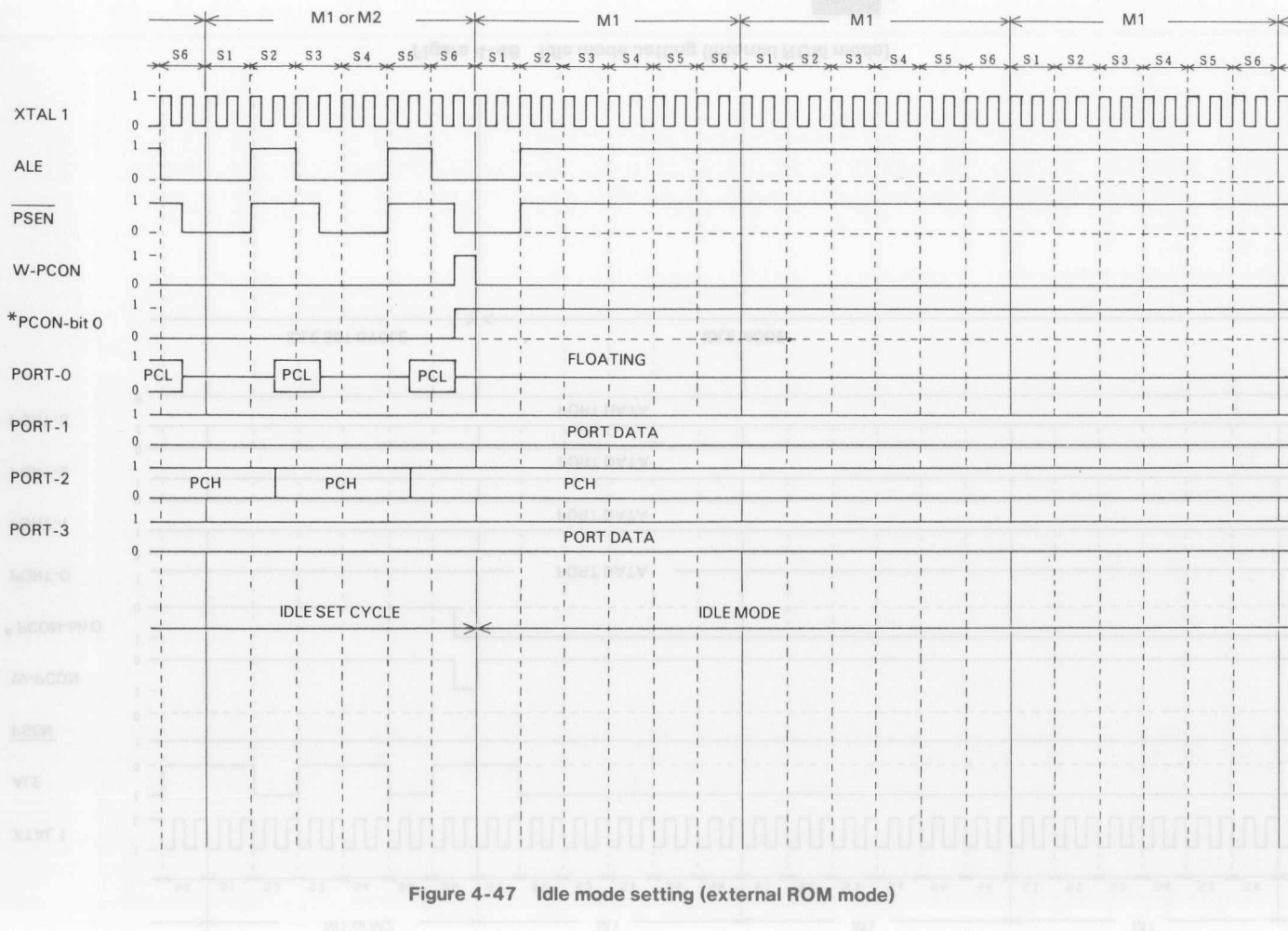


Figure 4-47 Idle mode setting (external ROM mode)

#### 4.8.3 Power down mode (PD) setting

Power down mode is set when "1" is set in the PD bit of the power control register (PCON, 87H). The circuit connection involved in this setting is shown in Figure 4-48.

In power down mode, both the XTAL1·2 and the CPU operations are halted.

The integrity of the CPU data memory (RAM) and I/O port data, however, is preserved during this mode.

The CPU pin status during the power down mode is outlined in Table 4-21, and the corresponding time charts for initiating the power down mode are shown in Figures 4-49 and 4-50.

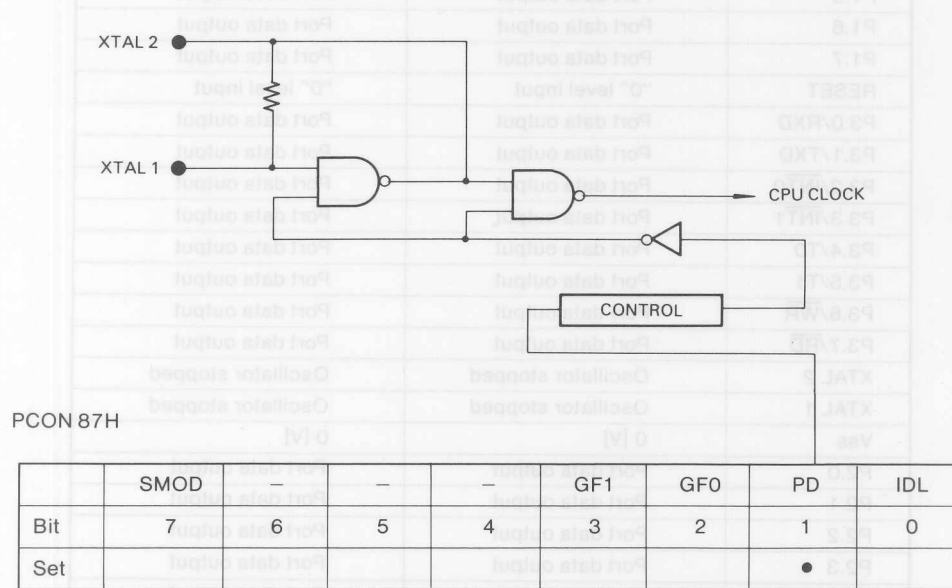


Figure 4-48 Power down mode equivalent circuit

Table 4-21 CPU pin status in power down mode

Name	Internal ROM	External ROM
P1.0	Port data output	Port data output
P1.1	Port data output	Port data output
P1.2	Port data output	Port data output
P1.3	Port data output	Port data output
P1.4	Port data output	Port data output
P1.5	Port data output	Port data output
P1.6	Port data output	Port data output
P1.7	Port data output	Port data output
RESET	"0" level input	"0" level input
P3.0/RXD	Port data output	Port data output
P3.1/TXD	Port data output	Port data output
P3.2/ $\overline{\text{INT0}}$	Port data output	Port data output
P3.3/ $\overline{\text{INT1}}$	Port data output	Port data output
P3.4/T0	Port data output	Port data output
P3.5/T1	Port data output	Port data output
P3.6/ $\overline{\text{WR}}$	Port data output	Port data output
P3.7/ $\overline{\text{RD}}$	Port data output	Port data output
XTAL 2	Oscillator stopped	Oscillator stopped
XTAL 1	Oscillator stopped	Oscillator stopped
Vss	0 [V]	0 [V]
P2.0	Port data output	Port data output
P2.1	Port data output	Port data output
P2.2	Port data output	Port data output
P2.3	Port data output	Port data output
P2.4	Port data output	Port data output
P2.5	Port data output	Port data output
P2.6	Port data output	Port data output
P2.7	Port data output	Port data output
PSEN	"0" level output	"0" level output
ALE	"0" level output	"0" level output
$\overline{\text{EA}}$	"1" level input	"0" level input
P0.7	Port data output	Floating
P0.6	Port data output	Floating
P0.5	Port data output	Floating
P0.4	Port data output	Floating
P0.3	Port data output	Floating
P0.2	Port data output	Floating
P0.1	Port data output	Floating
P0.0	Port data output	Floating
Vcc	+2.0 ~ +6 [V]	+2.0 ~ +6 [V]

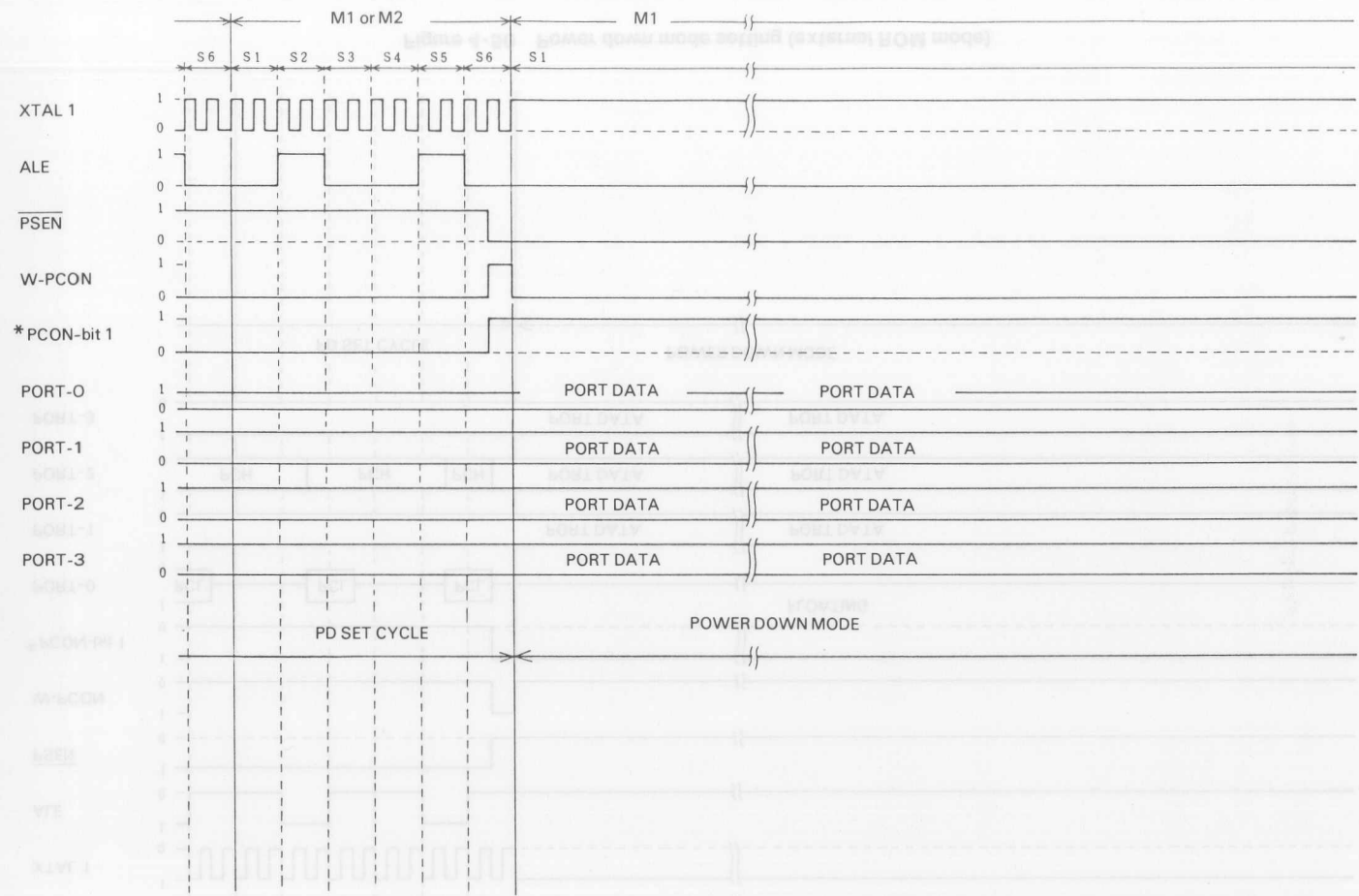


Figure 4-49 Power down mode setting (internal ROM mode)



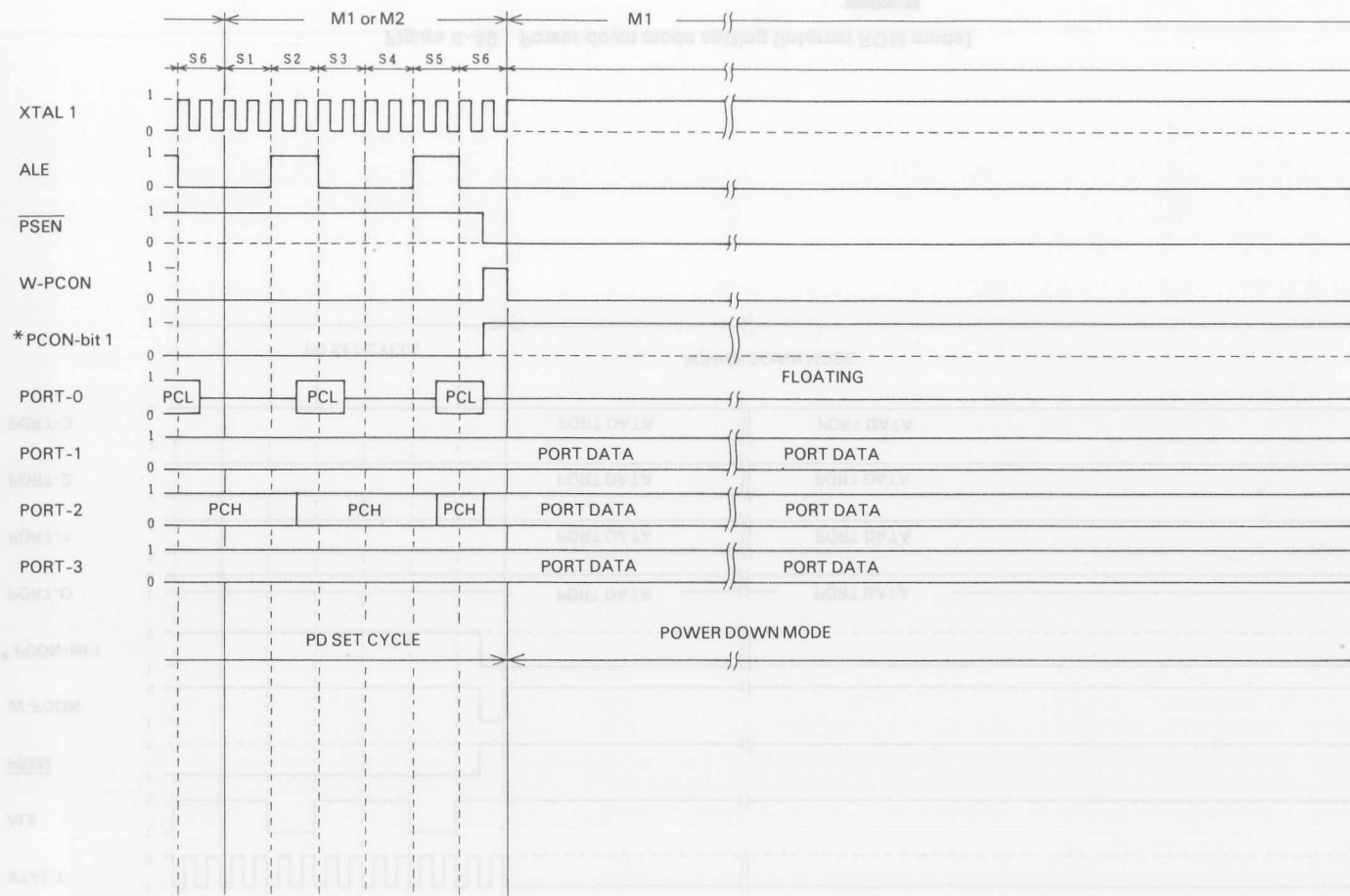


Figure 4-50 Power down mode setting (external ROM mode)

#### 4.9. CPU Power Down (IDLE and PD) Cancellation (CPU Activation)

### 4.9.1 Outline

CPU power down (IDLE and PD) can be cancelled (CPU activation) in two ways.

One way is to reset the CPU and execute from address 0 (common method for both IDLE and PD modes), and the other is to generate a CPU interrupt and execute from the interrupt address. This method can only be used if device is in the IDLE mode.

#### 4.9.2 Cancellation by CPU resetting

The CPU is reset when a “1” is applied to the CPU RESET pin, and the CPU power down state (IDLE or PD mode) is cancelled. Programs are subsequently executed by the CPU from address 0. The reset time charts are outlined in Figures 4-51 thru 4-54.

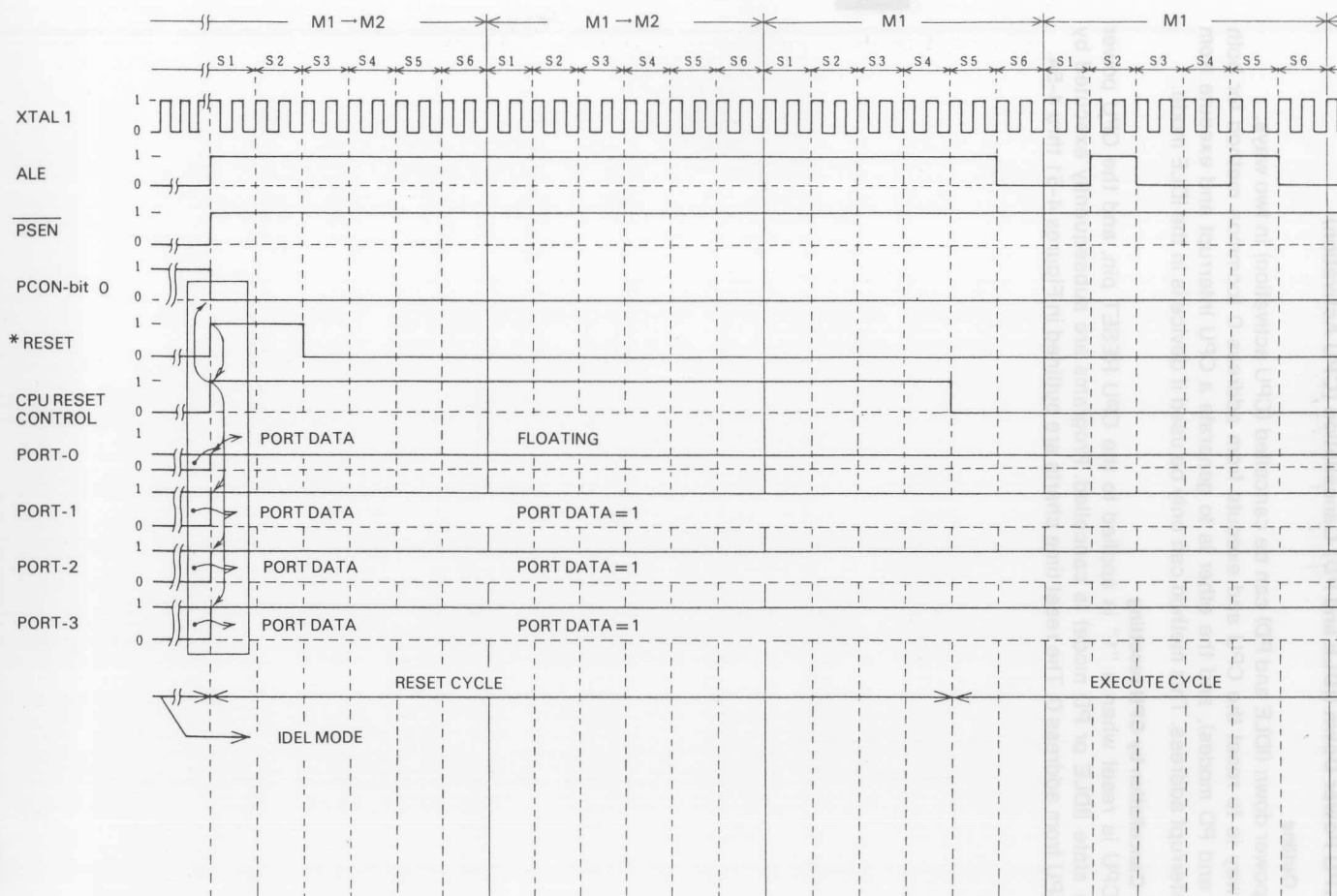


Figure 4-51 Restart from idle mode by resetting (internal ROM mode)

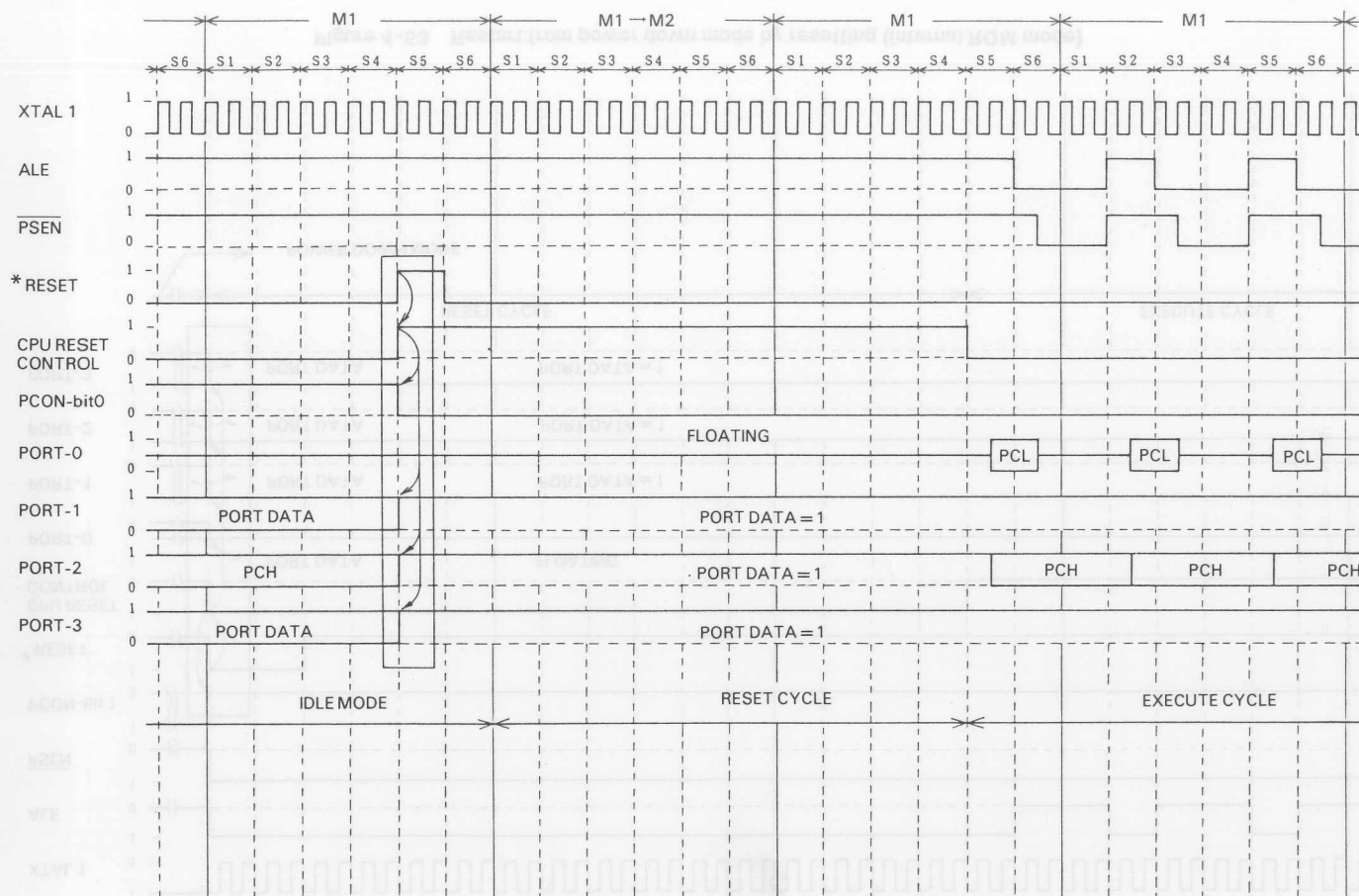


Figure 4-52 Restart from idle mode by resetting (external ROM mode)

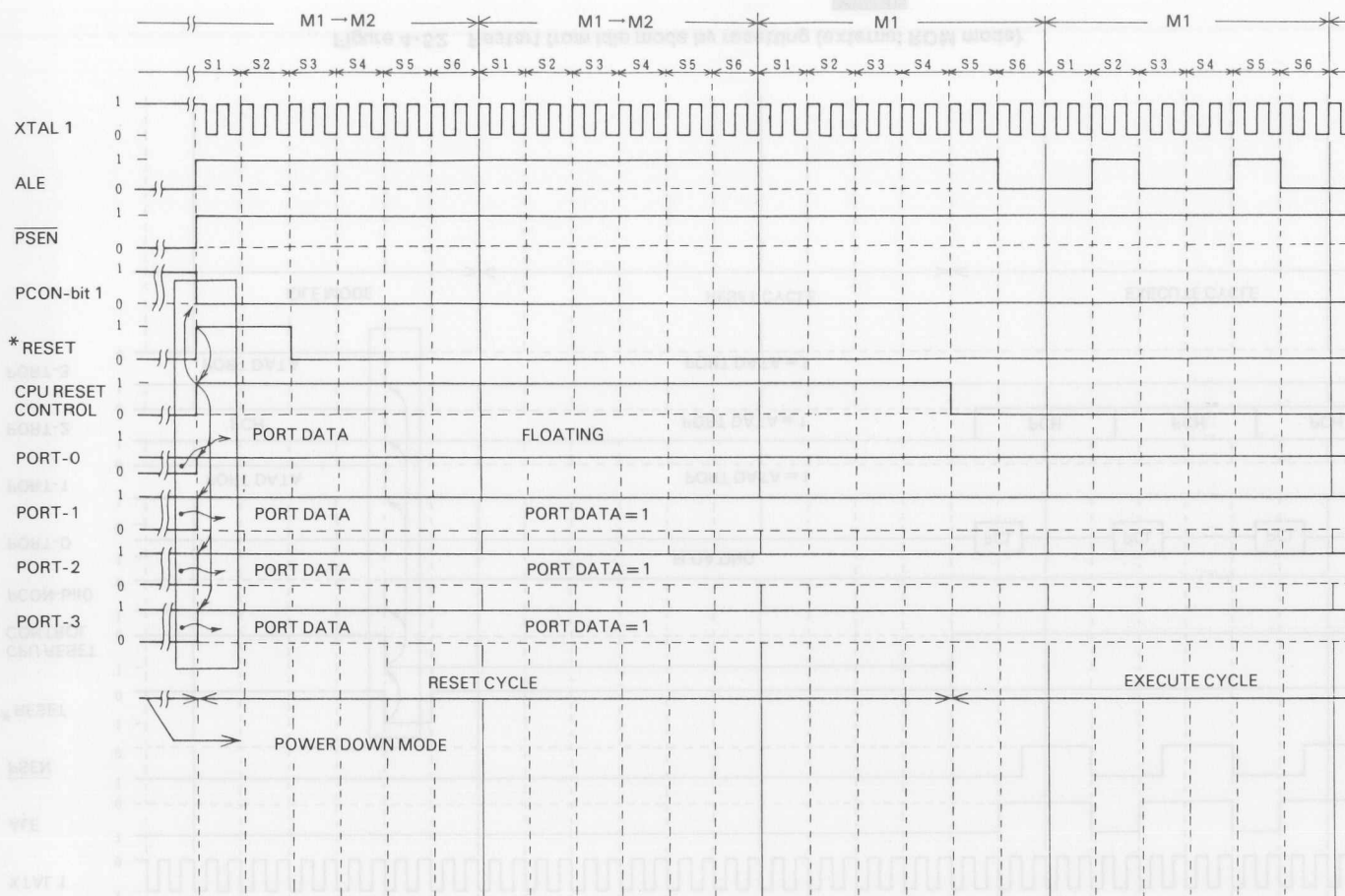


Figure 4-53 Restart from power down mode by resetting (internal ROM mode)

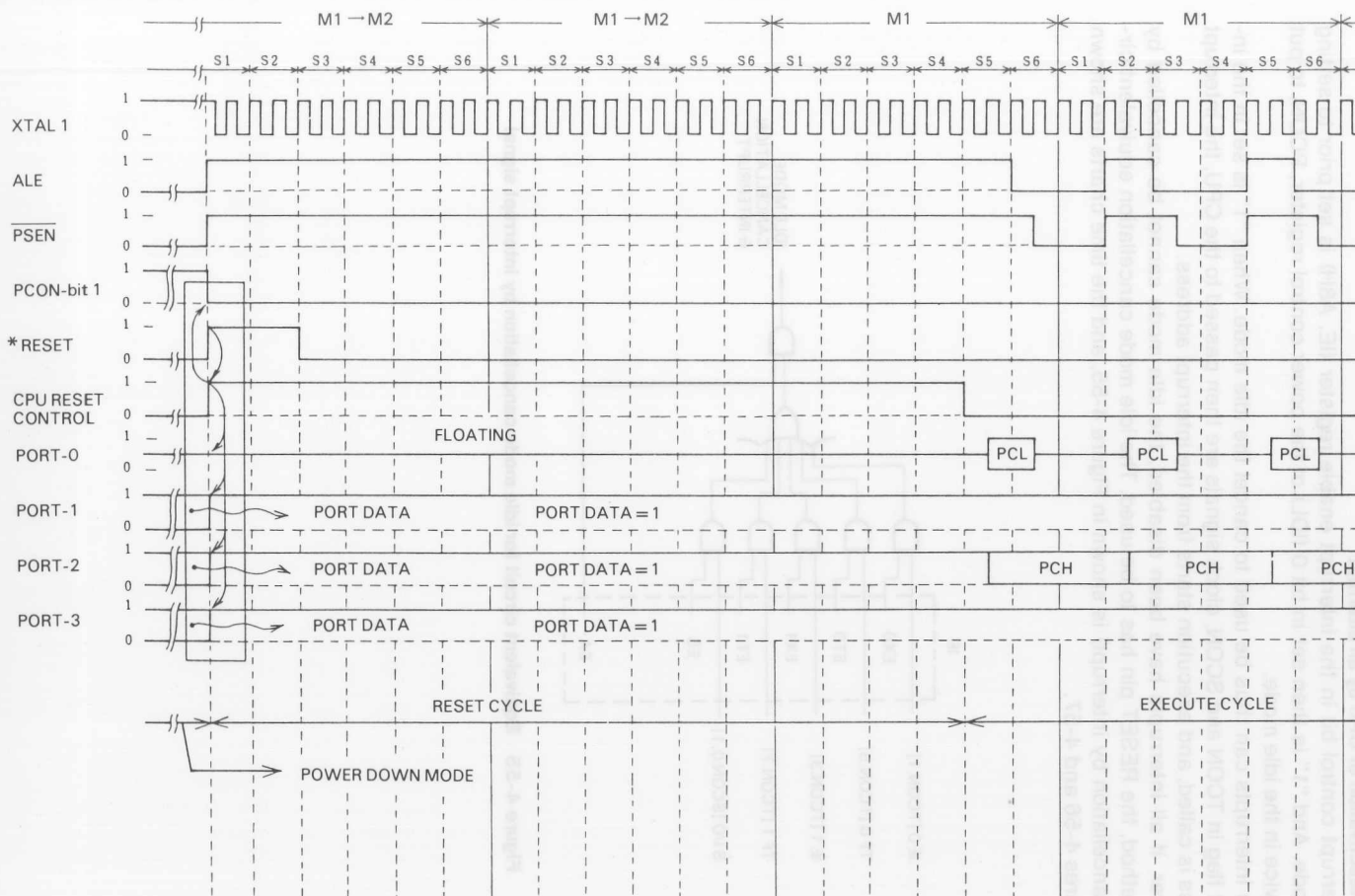


Figure 4-54 Restart from power down mode by resetting (external ROM mode)

#### 4.9.3 Reactivation of CPU by an interrupt

An interrupt control bit in the interrupt enable register (IE, A8H) is set prior to setting idle mode. And "1" is then set in bit 0 (IDL) of the power control register, PCON, to put the device in the idle mode.

All five interrupts can thus be used to cancel the idle mode. When "1" is set in the interrupt flag in TCON and SCON, clock signals are then passed to the CPU, the interrupt address is called, and execution starts from the interrupt address.

However, if all interrupts have been disabled, the idle mode cannot be cancelled by this method, the RESET pin has to be used. The idle mode cancellation equivalent circuit (cancellation by interrupt) is shown in Figure 4-55, and the time charts are shown in Figures 4-56 and 4-57.

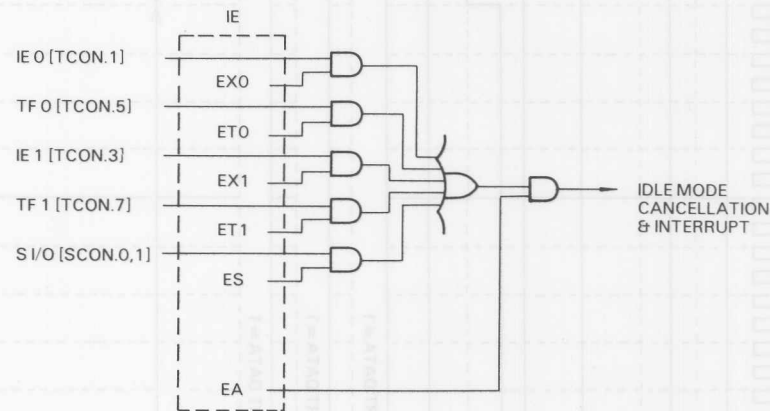


Figure 4-55 Equivalent circuit for idle mode cancellation by interrupt signal

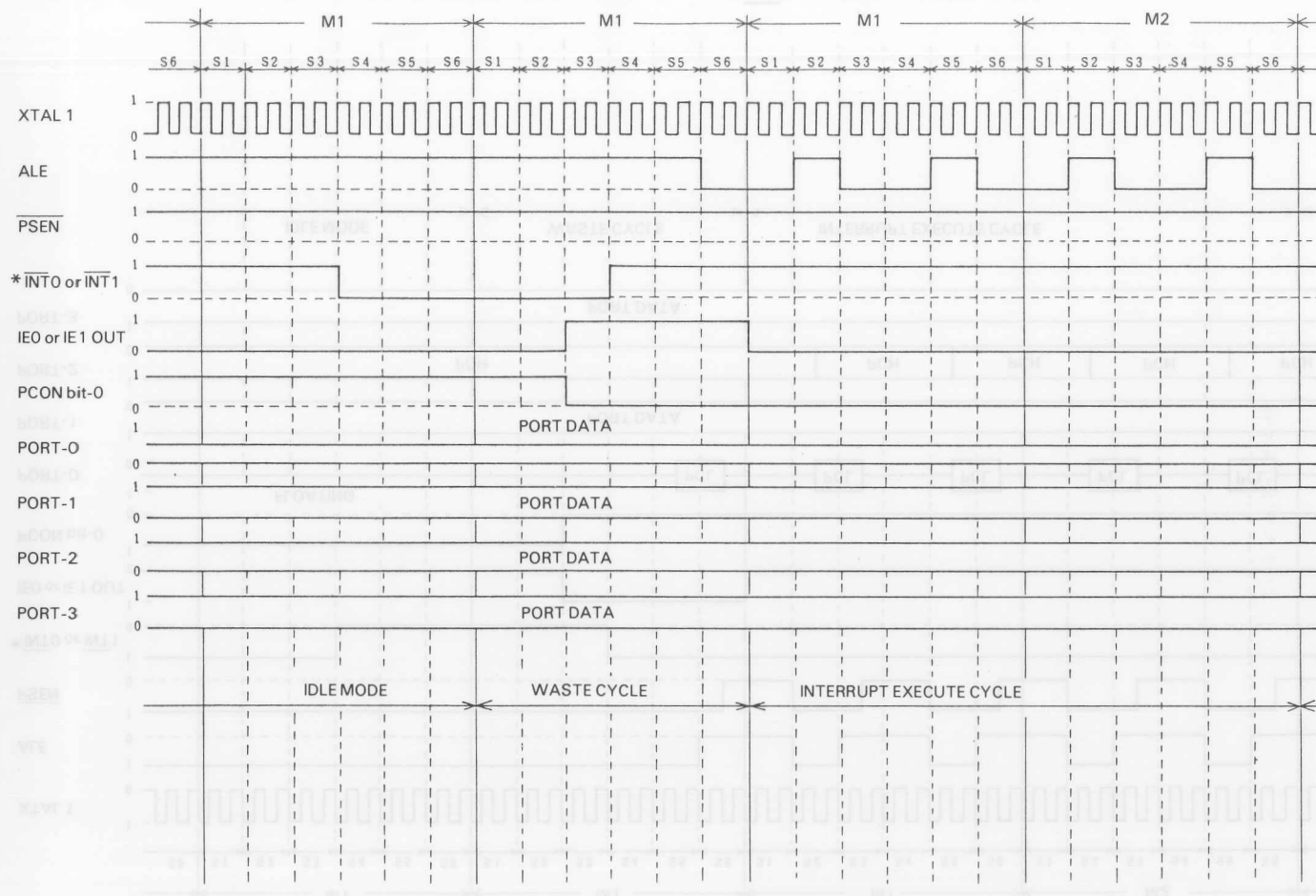
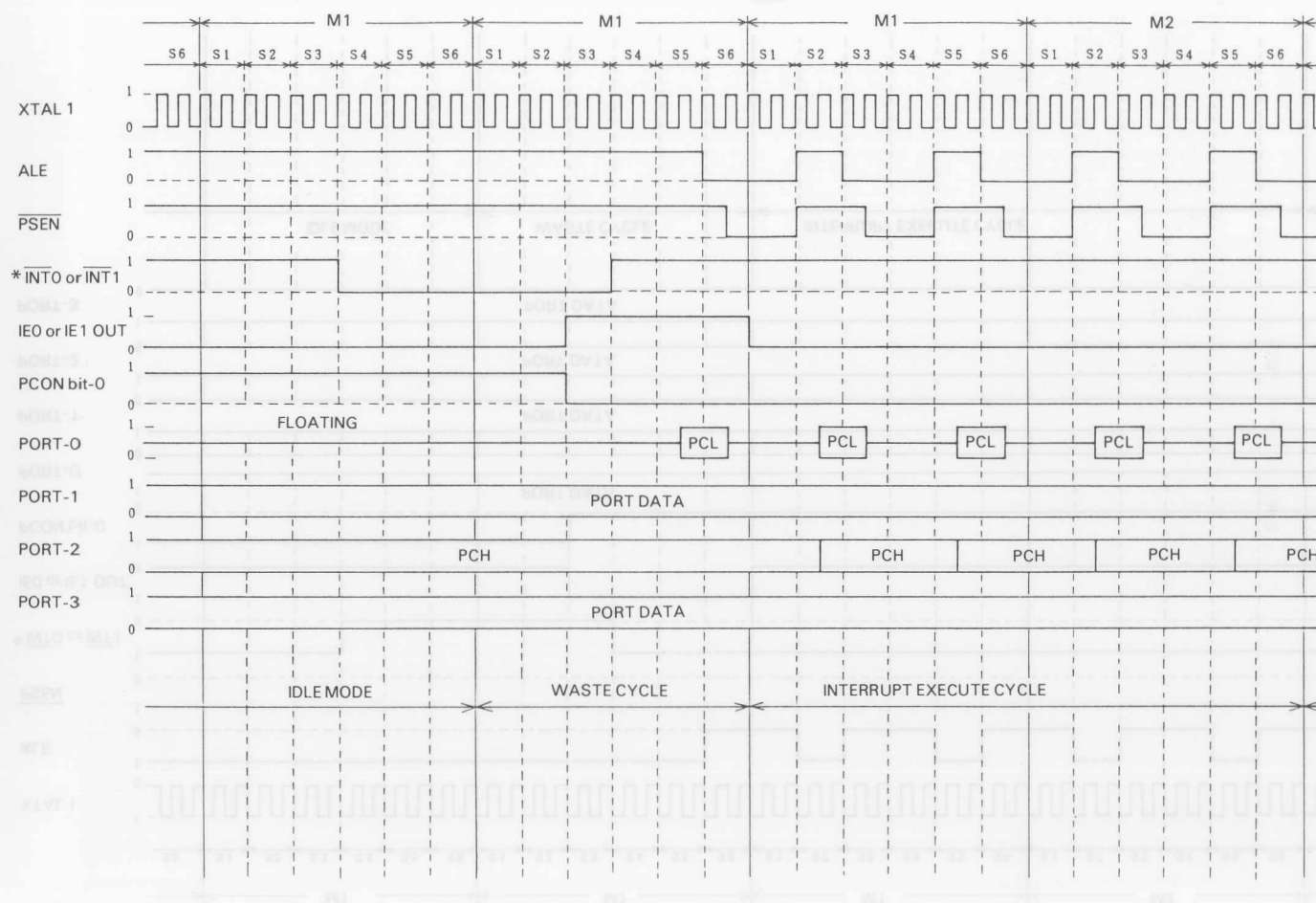


Figure 4-56 Restart from idle mode by interrupt  $\overline{\text{INT0}}$  or 1 (internal ROM mode)



Figure 4-57 Restart from idle mode by interrupt  $\overline{\text{INT0}}$  or 1 (external ROM mode)

## 5. INPUT/OUTPUT PORTS

# 5. INPUT/OUTPUT PORTS

## 5.2 Port 0

Port 0 is an 8-bit input/output port with circuit structure indicated in Figure 5-1. When port 0 is used as an input/output port in internal ROM mode (MEM80C31), the equivalent circuit of Port 0 is indicated in Figure 5-2. When operated as an output port, Port 0 becomes an open drain output port, and when operated as an input port, "1" has to be set in the port 0 latch to put the port 0 pin into floating status prior to using the port for input purposes.

When port 0 is used in external ROM mode (MEM80C31) and external RAM mode, the equivalent circuit is as shown in Figure 5-3 where addresses and data outputs are obtained as "1" and "0" by totem pole output driver. When data from external ROM or external RAM is input, port 0 automatically becomes a tri-state input port. When the CPU is reset or when an external ROM or external RAM is accessed, "1" is set automatically in the port 0 latch. The port 0 pin table is shown in Table 5-1.

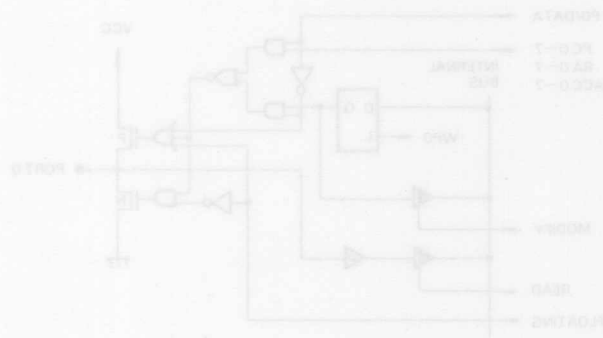


Figure 5-1 Port 0 internal equivalent circuit

## 5. INPUT/OUTPUT PORTS

## 5.1 Outline

MSM80C31/MSM80C51 is equipped with four 8-bit input/output ports. The functions of these four ports (port 0, 1, 2, and 3) are listed below.

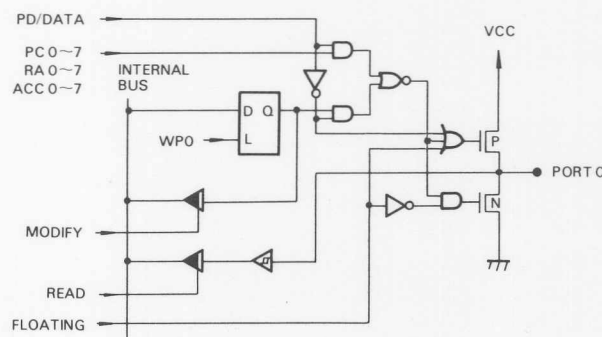
- 1) Port 0: Input/output bus port, address output port, and data input/output port.
- 2) Port 1: Quasi-bidirectional input/output port.
- 3) Port 2: Quasi-bidirectional input/output port and address output port.
- 4) Port 3: Quasi-bidirectional input/output port and control input/output pin.

## 5.2 Port 0

Port 0 is an 8-bit input/output port with circuit structure indicated in Figure 5-1. When port 0 is used as an input/output port in internal ROM mode (MSM80C51), the equivalent circuit of Port 0 is indicated in Figure 5-2. When operated as an output port, port 0 becomes an open drain output port, and when operated as an input port, “1” has to be set in the port 0 latch to put the port 0 pin into floating status prior to using the port for input purposes.

When port 0 is used in external ROM mode (MSM80C31) and external RAM mode, the equivalent circuit is as shown in Figure 5-3 where addresses and data outputs are obtained as “1” and “0” by totem pole output driver. When data from external ROM or external RAM is input, port 0 automatically becomes a tri-state input port.

When the CPU is reset or when an external ROM or external RAM is accessed, “1” is set automatically in the port 0 latch. The port 0 pin table is shown in Table 5-1.



**Figure 5-1 Port 0 internal equivalent circuit**

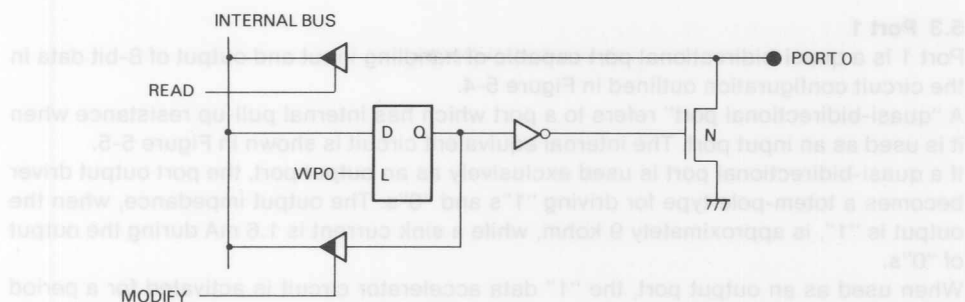


Figure 5-2 Port 0 input/output port equivalent circuit in internal ROM mode

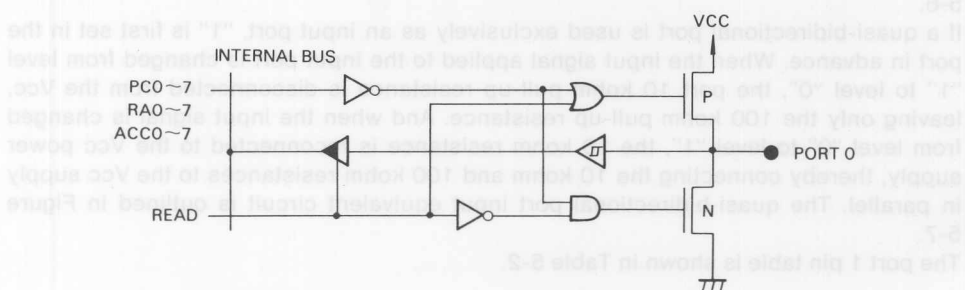


Figure 5-3 Port 0 equivalent circuit during address and data input/output when in external ROM/RAM mode

Table 5-1 Port 0 pin table

	PORT 0	Accumulator bit	Address
1	P0.0	ACC.0	PC-0 RA-0
2	P0.1	ACC.1	PC-1 RA-1
3	P0.2	ACC.2	PC-2 RA-2
4	P0.3	ACC.3	PC-3 RA-3
5	P0.4	ACC.4	PC-4 RA-4
6	P0.5	ACC.5	PC-5 RA-5
7	P0.6	ACC.6	PC-6 RA-6
8	P0.7	ACC.7	PC-7 RA-7

### 5.3 Port 1

Port 1 is a quasi-bidirectional port capable of handling input and output of 8-bit data in the circuit configuration outlined in Figure 5-4.

A "quasi-bidirectional port" refers to a port which has internal pull-up resistance when it is used as an input port. The internal equivalent circuit is shown in Figure 5-5.

If a quasi-bidirectional port is used exclusively as an output port, the port output driver becomes a totem-pole type for driving "1"s and "0"s. The output impedance, when the output is "1", is approximately 9 kohm, while a sink current is 1.6 mA during the output of "0"s.

When used as an output port, the "1" data accelerator circuit is activated for a period equivalent to two XTAL1/2 oscillator clocks only when the output data is shifted from "0" to "1". During this data acceleration operation, the "1" output impedance is changed to about 500 ohms, the IOH current is increased, and the output signal's leading edge is speeded up. The accelerator circuit operation time chart is shown in Figure 5-6.

If a quasi-bidirectional port is used exclusively as an input port, "1" is first set in the port in advance. When the input signal applied to the input port is changed from level "1" to level "0", the port 10 kohm pull-up resistance is disconnected from the Vcc, leaving only the 100 kohm pull-up resistance. And when the input signal is changed from level "0" to level "1", the 10 kohm resistance is reconnected to the Vcc power supply, thereby connecting the 10 kohm and 100 kohm resistances to the Vcc supply in parallel. The quasi-bidirectional port input equivalent circuit is outlined in Figure 5-7.

The port 1 pin table is shown in Table 5-2.

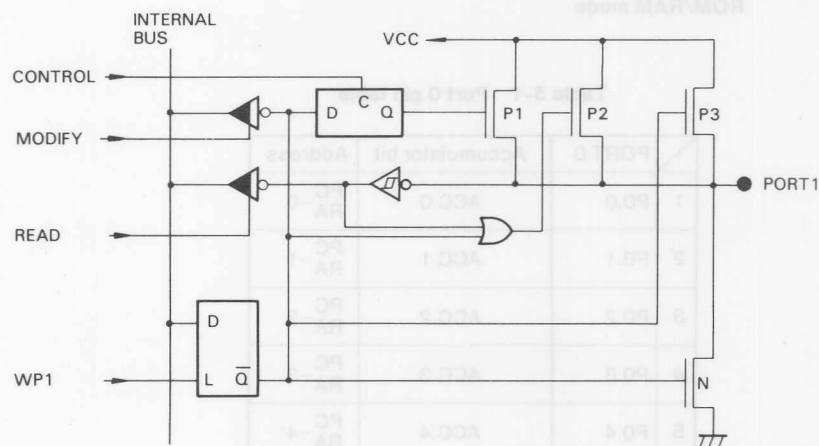


Figure 5-4 Port 1 internal equivalent circuit

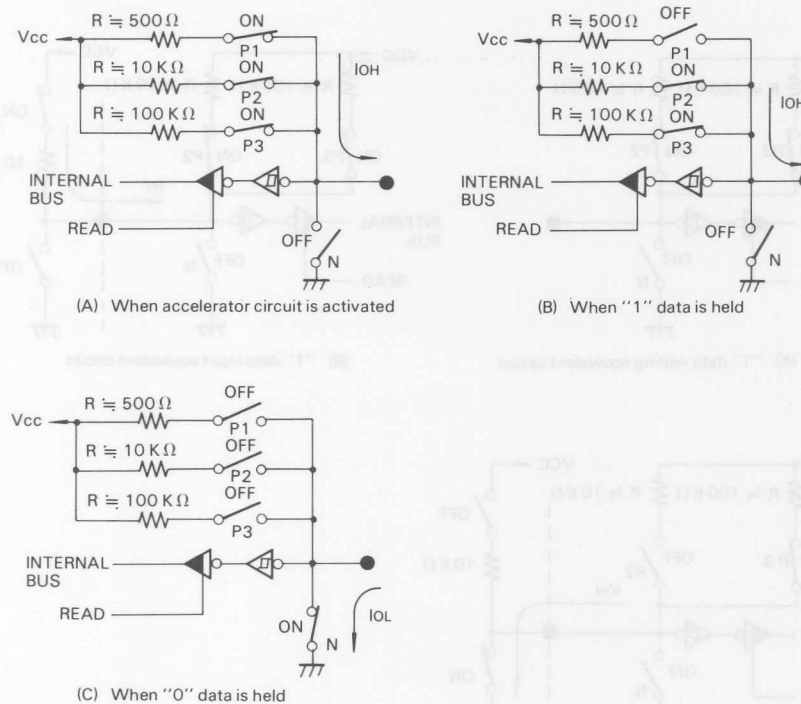


Figure 5-5 Quasi-bidirectional port equivalent circuit

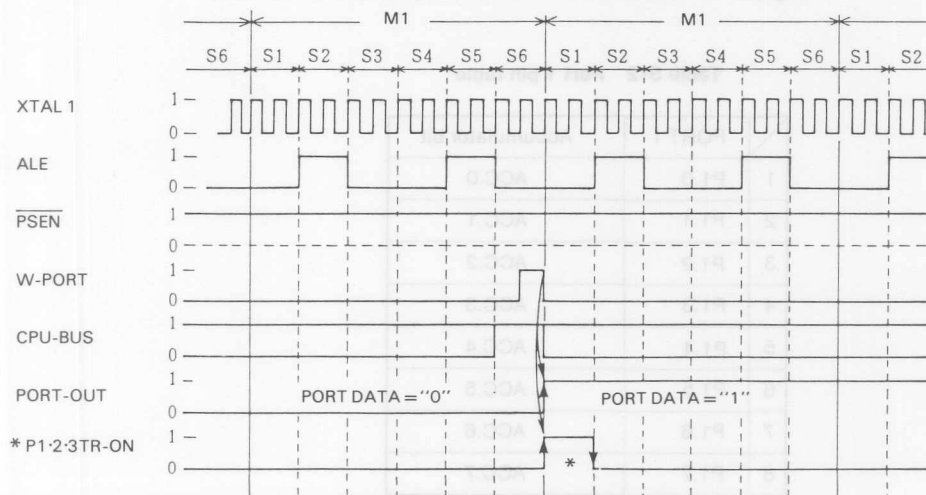


Figure 5-6 Quasi-bidirectional port accelerator circuit operation time chart

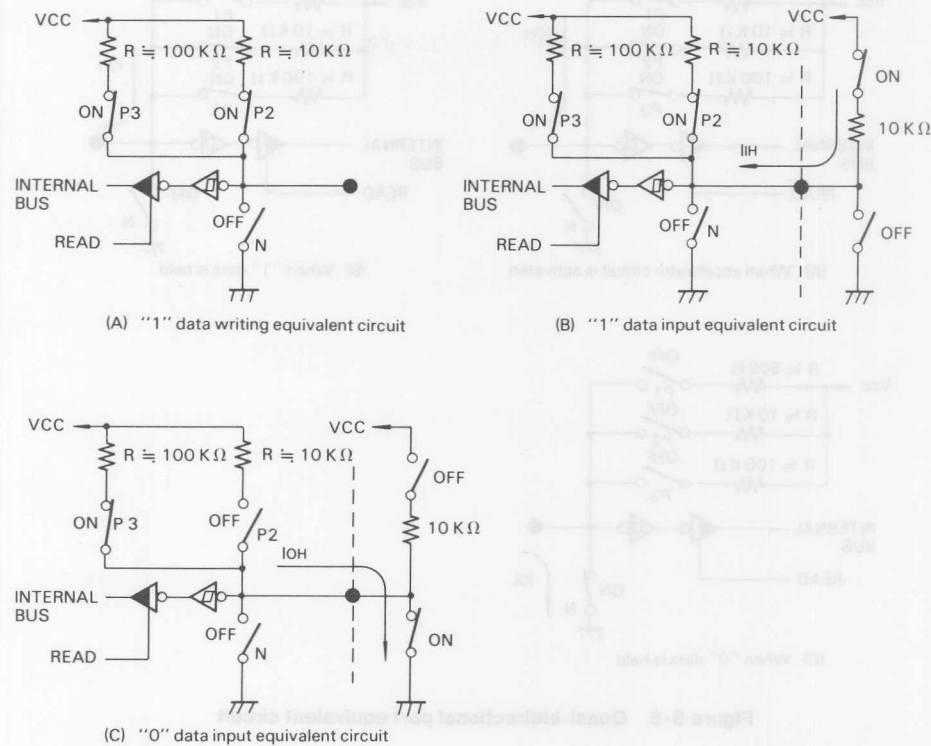


Figure 5-7 Quasi-bidirectional port input equivalent circuit

Table 5-2 Port 1 pin table

	PORT 1	Accumulator bit
1	P1.0	ACC.0
2	P1.1	ACC.1
3	P1.2	ACC.2
4	P1.3	ACC.3
5	P1.4	ACC.4
6	P1.5	ACC.5
7	P1.6	ACC.6
8	P1.7	ACC.7

### 5.4 Port 2

Port 2 can function as a quasi-bidirectional port capable of handling input and output of 8-bit data in the circuit configuration outlined in Figure 5-8. It can also be used for output of addresses 8 thru 15 in external ROM mode and external RAM mode using the data pointer DPTR.

When port 2 is used as a quasi-bidirectional port, it functions in much the same way as port 1. Note, however, that the port 2 "1" data accelerator circuit operates for a period equivalent to four XTAL1·2 oscillator clocks.

Output of addresses 8 thru 15 obtained from port 2 makes use of the circuit outlined in Figure 5-9. When the address output data is "1", the "1" data accelerator circuit is activated during the output of the data, resulting in a larger drive capability.

The port 2 pin table is given in Table 5-3.

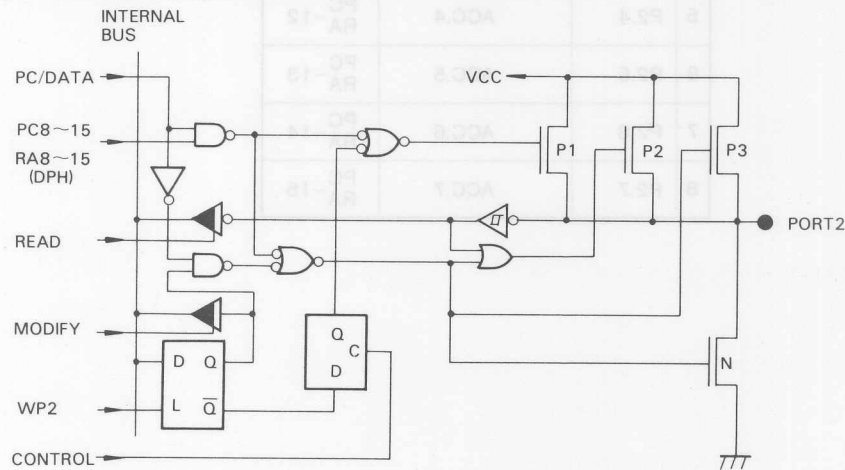


Figure 5-8 Port 2 internal equivalent circuit

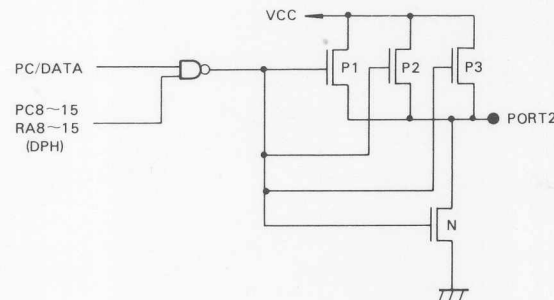


Figure 5-9 Port 2 ROM/RAM address output equivalent circuit



Table 5-3 Port 2 pin table

	PORT 2	Accumulator bit	Address
1	P2.0	ACC.0	PC RA-8
2	P2.1	ACC.1	PC RA-9
3	P2.2	ACC.2	PC RA-10
4	P2.3	ACC.3	PC RA-11
5	P2.4	ACC.4	PC RA-12
6	P2.5	ACC.5	PC RA-13
7	P2.6	ACC.6	PC RA-14
8	P2.7	ACC.7	PC RA-15

quasi-bidirectional port all functions a

quasi-bidirectional port, all functions are  
then used as a GPI control via the

in Table 5-5.



### 5-10 Port 3 internal equivalent circuit

Table 5-4 Port 3 alternate function table

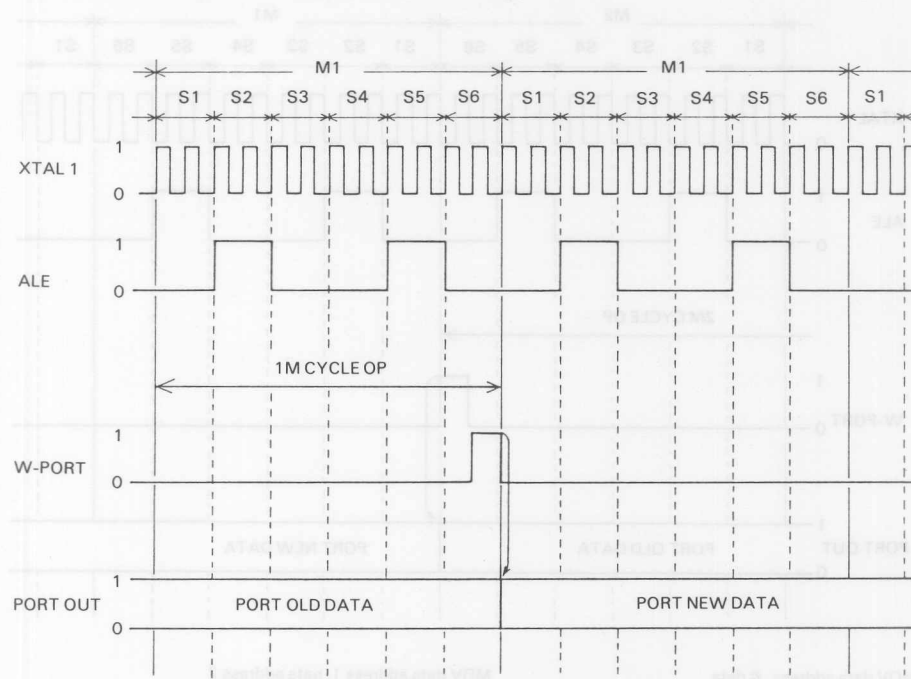
PORT 3	PORT 3 PIN ALTERNATE FUNCTION
P3.0	RXD [SERIAL INPUT PORT]
P3.1	TXD [SERIAL OUTPUT PORT]
P3.2	$\overline{\text{INT0}}$ [EXTERNAL INTERRUPT 0]
P3.3	$\overline{\text{INT1}}$ [EXTERNAL INTERRUPT 1]
P3.4	T0 [TIMER/COUNTER 0 CLOCK]
P3.5	T1 [TIMER/COUNTER 1 CLOCK]
P3.6	$\overline{\text{WR}}$ [EXTERNAL DATA MEMORY WRITE STROBE]
P3.7	$\overline{\text{RD}}$ [EXTERNAL DATA MEMORY READ STROBE]

Table 5-5 Port 3 pin table

	PORT 3	Control	Accumulator bit
1	P3.0	RXD	ACC.0
2	P3.1	TXD	ACC.1
3	P3.2	$\overline{\text{INT0}}$	ACC.2
4	P3.3	$\overline{\text{INT1}}$	ACC.3
5	P3.4	T0	ACC.4
6	P3.5	T1	ACC.5
7	P3.6	$\overline{\text{WR}}$	ACC.6
8	P3.7	$\overline{\text{RD}}$	ACC.7

## 5.6 Port Output Timing

### 1) One machine cycle instruction output timing



INC data address

DEC data address

MOV data address, A

ORL data address, A

ANL data address, A

XRL data address, A

XCH A, data address

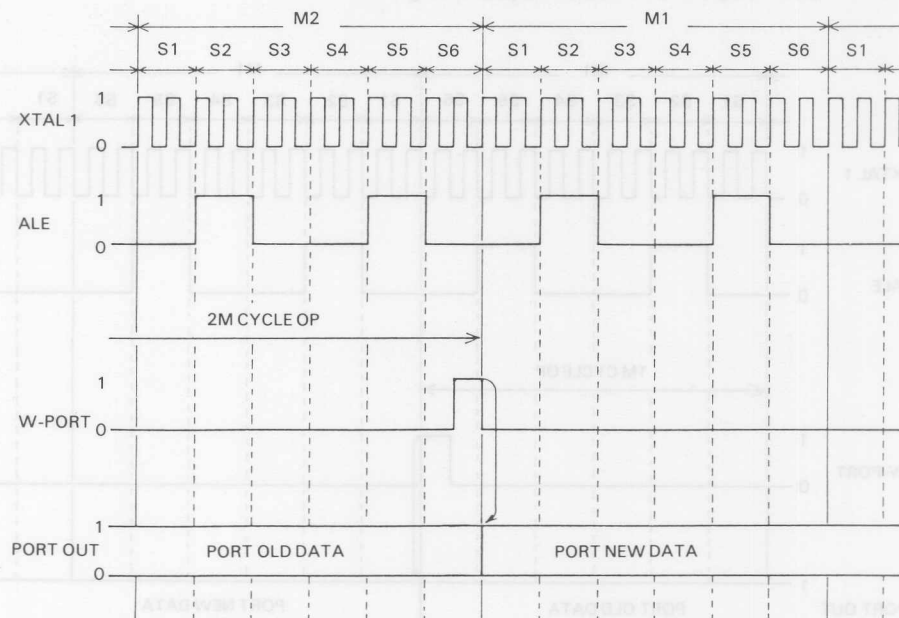
CPL bit address

CLR bit address

SETB bit address

Figure 5-11 One machine cycle instruction port output time chart

2) Two machine cycle instruction output timing



MOV data address, # data

MOV data address 1, oata address 2

ORL data address, # data

MOV bit address, C

ANL data address, # data

XRL data address, # data

JBC bit address, Code address

POP data address

MOV data address, @Rr

MOV data address, Rr

Figure 5-12 Two machine cycle instruction port output time chart

### 5.7 Port Data Manipulating Instructions

The MSM80C31/MSM80C51 port operation instructions for ports 0, 1, 2, and 3 are divided into two groups – one where external signals applied to the port pin are used according to the instruction to be executed, and the other where port latch data unaffected by the applied external signals is used. Instructions which use port latch data are listed below.

INC	data address
DEC	data address
ORL	data address, # data
ANL	data address, # data
XRL	data address, # data
ORL	data address, A
ANL	data address, A
XRL	data address, A
CPL	bit address
JBC	bit address, Code address
DJNZ	data address, Code address
PUSH	data address



## 6. MSM80C31/MSM80C51 ELECTRICAL CHARACTERISTICS

# 6. MSM80C31/ MSM80C51 ELECTRICAL CHARACTERISTICS

Unit	Rating	Condition
V	$-0.5 \sim 7$	$-55^{\circ}\text{C} \sim +25^{\circ}\text{C}$
V	$-0.5 \sim V_{CC} + 0.5$	$-55^{\circ}\text{C} \sim +25^{\circ}\text{C}$
$^{\circ}\text{C}$	$-55 \sim +150$	

Unit	Rating	Condition	Symbol	Parameter
V	$2.5 \sim 8$	$f = \text{fosc} = 0\text{C} \sim 16\text{ MHz}$	$V_{CC}$	Operating voltage
V	$2 \sim 8$		$V_{CC}$	Memory hold voltage
$^{\circ}\text{C}$	$-40 \sim +85$		$T_A$	Ambient temperature

1. DC and AC characteristics in the range of  $12\text{ MHz} < f \leq 16\text{ MHz}$  and  $2.5\text{ V} \leq V_{CC} < 4\text{ V}$  will be specified elsewhere.



## 6. MSM80C31/MSM80C51 ELECTRICAL CHARACTERISTICS

### 6.1 Absolute Maximum Ratings

Parameter	Symbol	Condition	Rating	Unit
Supply voltage	V <sub>CC</sub>	T <sub>a</sub> = 25°C	- 0.5 ~ 7	V
Input voltage	V <sub>I</sub>	T <sub>a</sub> = 25°C	- 0.5 ~ V <sub>CC</sub> + 0.5	V
Storage temperature	T <sub>stg</sub>		-55 ~ +150	°C

### 6.2 Operational Ranges

Parameter	Symbol	Condition	Rating	Unit
Operating voltage	V <sub>CC</sub>	*1 f <sub>osc</sub> =DC~16 MHz	2.5 ~ 6	V
Memory hold voltage	V <sub>CC</sub>		2 ~ 6	V
Ambient temperature	T <sub>a</sub>		-40 ~ +85	°C

\*1 DC and AC characteristics in the range of  $12 \text{ MHz} < f \leq 16 \text{ MHz}$  and  $2.5 \text{ V} \leq V_{CC} < 4 \text{ V}$  will be specified elsewhere.

# MSM80C31/MSM80C51 ELECTRICAL CHARACTERISTICS

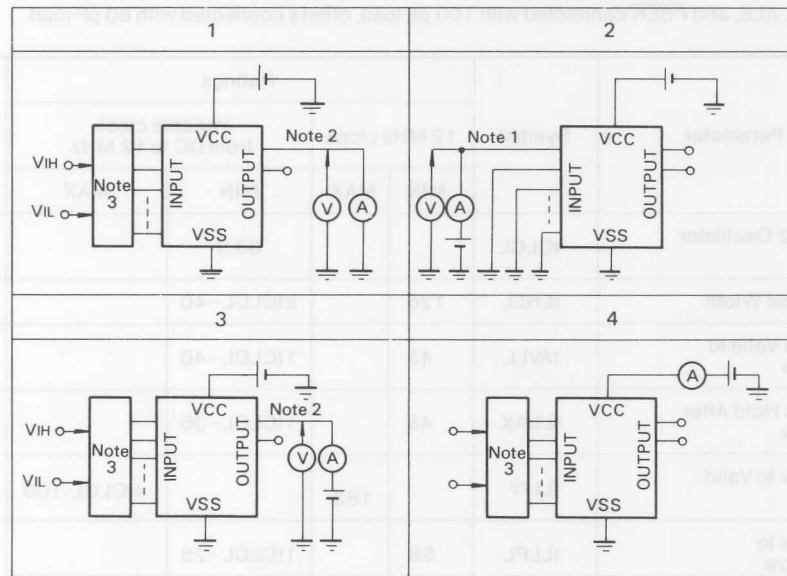
## 6.3 DC Characteristics

( $V_{CC} = 5\text{ V} \pm 20\%$ ,  $V_{SS} = 0\text{ V}$ ,  $T_a = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ )

Parameter	Symbol	Conditions	MIN	TYP	MAX	Unit	Measuring Circuit
Input Low Voltage	$V_{IL}$		-0.5		$0.2 V_{CC} - 0.1$	V	1
Input High Voltage	$V_{IH}$	Except XTAL1 and RESET	$0.2 V_{CC} + 0.9$		$V_{CC} + 0.5$	V	
Input High Voltage	$V_{IH1}$	XTAL1 and RESET	$0.7 V_{CC}$		$V_{CC} + 0.5$	V	
Output Low Voltage [PORT 1, 2, 3]	$V_{OL}$	$I_{OL} = 1.6\text{ mA}$			0.45	V	
Output Low Voltage [PORT 0, ALE, PSEN]	$V_{OL1}$	$I_{OL} = 3.2\text{ mA}$			0.45	V	
Output High Voltage [PORT 1, 2, 3]	$V_{OH}$	$I_{OH} = -60\text{ }\mu\text{A}$ $V_{CC} = 5\text{ V} \pm 10\%$	2.4			V	
		$I_{OH} = -30\text{ }\mu\text{A}$	$0.75 V_{CC}$			V	
		$I_{OH} = -10\text{ }\mu\text{A}$	$0.9 V_{CC}$			V	
Output High Voltage [PORT 0, ALE, PSEN]	$V_{OH1}$	$I_{OH} = -400\text{ }\mu\text{A}$ $V_{CC} = 5\text{ V} \pm 10\%$	2.4			V	
		$I_{OH} = -150\text{ }\mu\text{A}$	$0.75 V_{CC}$			V	
		$I_{OH} = -40\text{ }\mu\text{A}$	$0.9 V_{CC}$			V	
Logical 0 Input Current [PORT 1, 2, 3]	$I_{IL}$	$V_I = 0.45\text{ V}$			-200	$\mu\text{A}$	2
Logical 1 to 0 Transition Current [PORT 1, 2, 3]	$I_{TL}$	$V_I = 2.0\text{ V}$			-500	$\mu\text{A}$	
Input Leakage Current [PORT 0 floating, EA]	$I_{LI}$	$V_{SS} < V_I < V_{CC}$			$\pm 10$	$\mu\text{A}$	3
RESET Pulldown Resistor	$R_{RST}$		20	40	125	$\text{K}\Omega$	2
Pin Capacitance	$C_{IO}$	$T_a = 25^\circ\text{C}$ , $f = 1\text{ MHz}$ $5\text{ V}$ [except XTAL1]			10	pF	

Parameter	Symbol	Conditions	MIN	TYP	MAX	Unit	Measuring Circuit
Power Down Current	IPD	VCC = 2 V		1	50	$\mu$ A	
Maximum Power Supply Current Normal Operation	ICC	VCC =	4	5	6	V	
		fosc = 12 MHz	12	16	20	mA	
		fosc = 8 MHz	8.3	11	14	mA	
		fosc = 3.5 MHz	4.3	5.7	7.5	mA	4
		fosc = 0.5 MHz	1.6	2.2	3	mA	
Maximum Power Supply Current Idle Mode	ICC1	fosc = 12 MHz	2.5	3.7	5	mA	
		fosc = 8 MHz	1.8	2.7	3.7	mA	
		fosc = 3.5 MHz	1.1	1.6	2.2	mA	
		fosc = 0.5 MHz	0.6	0.9	1.2	mA	

## Measuring circuits



- Note 1. Repeated for specified input pins  
 Note 2. Repeated for specified output pins  
 Note 3. Input logic for specified status.

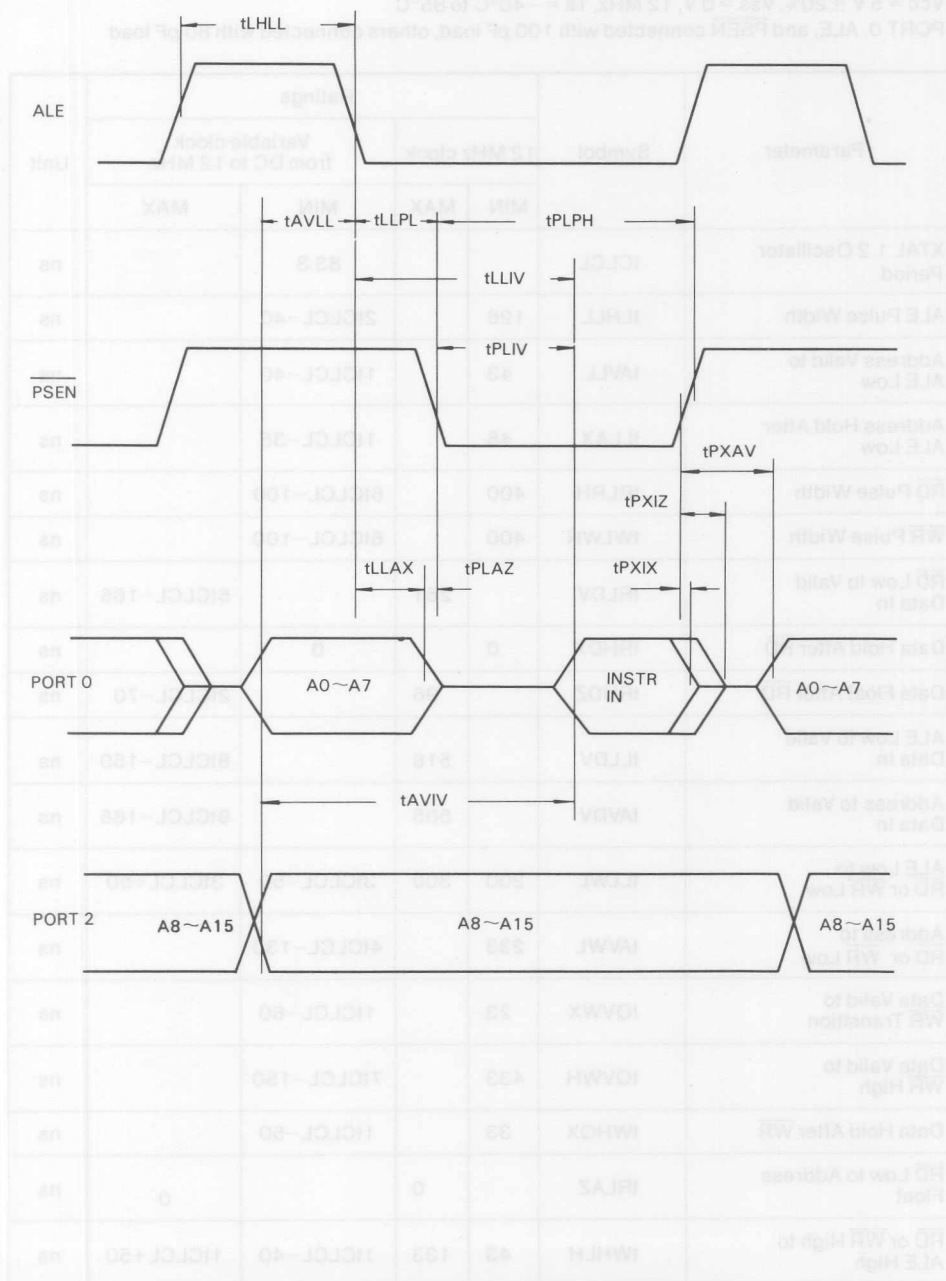
## 6.4 External Program Memory Access AC Characteristics

$V_{CC} = 5\text{ V} \pm 20\%$ ,  $V_{SS} = 0\text{ V}$ , 12 MHz,  $T_a = -40^\circ\text{C}$  to  $85^\circ\text{C}$

PORT 0, ALE, and  $\overline{\text{PSEN}}$  connected with 100 pF load, others connected with 80 pF load

Parameter	Symbol	Ratings				Unit
		12 MHz clock		Variable clock from DC to 12 MHz		
		MIN	MAX	MIN	MAX	
XTAL 1.2 Oscillator Period	tCLCL			83.3		ns
ALE Pulse Width	tLHLL	126		2tCLCL−40		ns
Address Valid to ALE Low	tAVLL	43		1tCLCL−40		ns
Address Hold After ALE Low	tLLAX	48		1tCLCL−35		ns
ALE Low to Valid Instr In	tLLIV		183		4tCLCL-100	ns
ALE Low to PSEN Low	tLLPL	58		1tCLCL−25		ns
PSEN Pulse Width	tPLPH	215		3tCLCL−35		ns
PSEN Low to Valid Instr In	tPLIV		100		3tCLCL−105	ns
Input Instr Hold After PSEN	tPXIX	0		0		ns
Input Instr Float After PSEN	tPXIZ		63		1tCLCL−20	ns
PSEN to Address Valid	tPXAV	75		1tCLCL−8		ns
Address to Valid Instr In	tAVIV		266		5tCLCL−105	ns
PSEN Low to Address Float	tPLAZ		0		0	ns

## External program memory read cycle



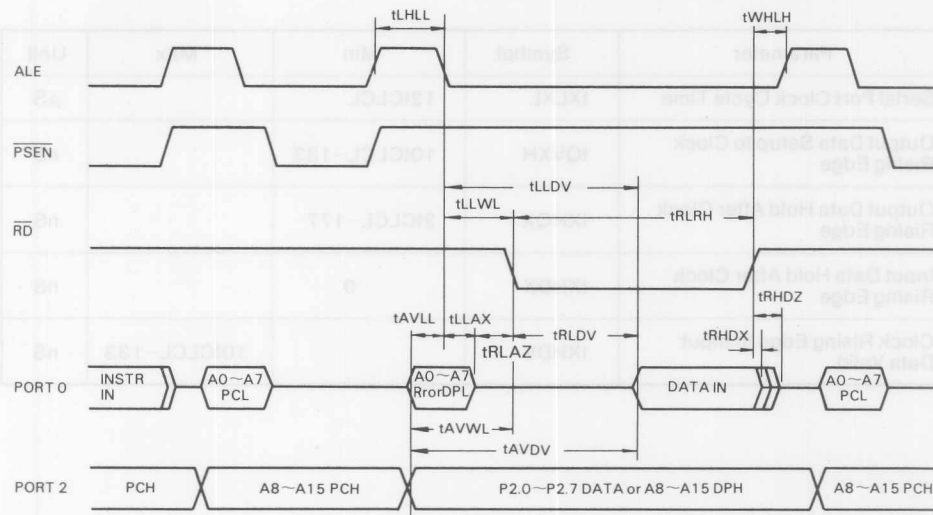
## 6.5 External Data Memory Access AC Characteristics

$V_{CC} = 5\text{ V} \pm 20\%$ ,  $V_{SS} = 0\text{ V}$ , 12 MHz,  $T_a = -40^\circ\text{C}$  to  $85^\circ\text{C}$

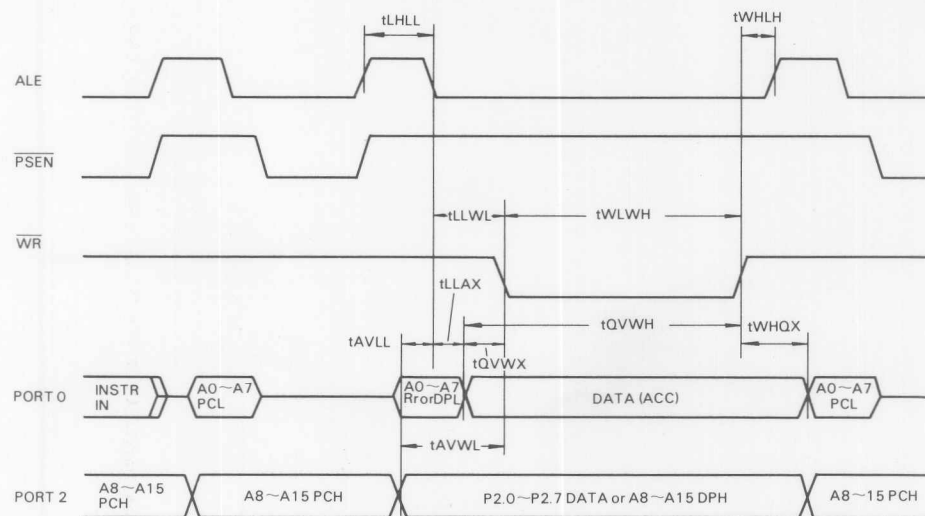
PORT 0, ALE, and PSEN connected with 100 pF load, others connected with 80 pF load

Parameter	Symbol	Ratings				Unit
		12 MHz clock		Variable clock from DC to 12 MHz		
		MIN	MAX	MIN	MAX	
XTAL 1.2 Oscillator Period	tCLCL			83.3		ns
ALE Pulse Width	tLHLL	126		2tCLCL−40		ns
Address Valid to ALE Low	tAVLL	43		1tCLCL−40		ns
Address Hold After ALE Low	tLLAX	48		1tCLCL−35		ns
$\overline{\text{RD}}$ Pulse Width	tRLRH	400		6tCLCL−100		ns
$\overline{\text{WR}}$ Pulse Width	tWLWH	400		6tCLCL−100		ns
$\overline{\text{RD}}$ Low to Valid Data In	tRLDV		251		5tCLCL−165	ns
Data Hold After $\overline{\text{RD}}$	tRHDX	0		0		ns
Data Float After $\overline{\text{RD}}$	tRHDZ		96		2tCLCL−70	ns
ALE Low to Valid Data In	tLLDV		516		8tCLCL−150	ns
Address to Valid Data In	tAVDV		585		9tCLCL−165	ns
ALE Low to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low	tLLWL	200	300	3tCLCL−50	3tCLCL+50	ns
Address to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low	tAVWL	233		4tCLCL−130		ns
Data Valid to $\overline{\text{WR}}$ Transition	tQVWX	23		1tCLCL−60		ns
Data Valid to $\overline{\text{WR}}$ High	tQVWH	433		7tCLCL−150		ns
Data Hold After $\overline{\text{WR}}$	tWHQX	33		1tCLCL−50		ns
$\overline{\text{RD}}$ Low to Address Float	tRLAZ		0		0	ns
$\overline{\text{RD}}$ or $\overline{\text{WR}}$ High to ALE High	tWHLH	43	133	1tCLCL−40	1tCLCL+50	ns

External data memory read cycle



External data memory write cycle



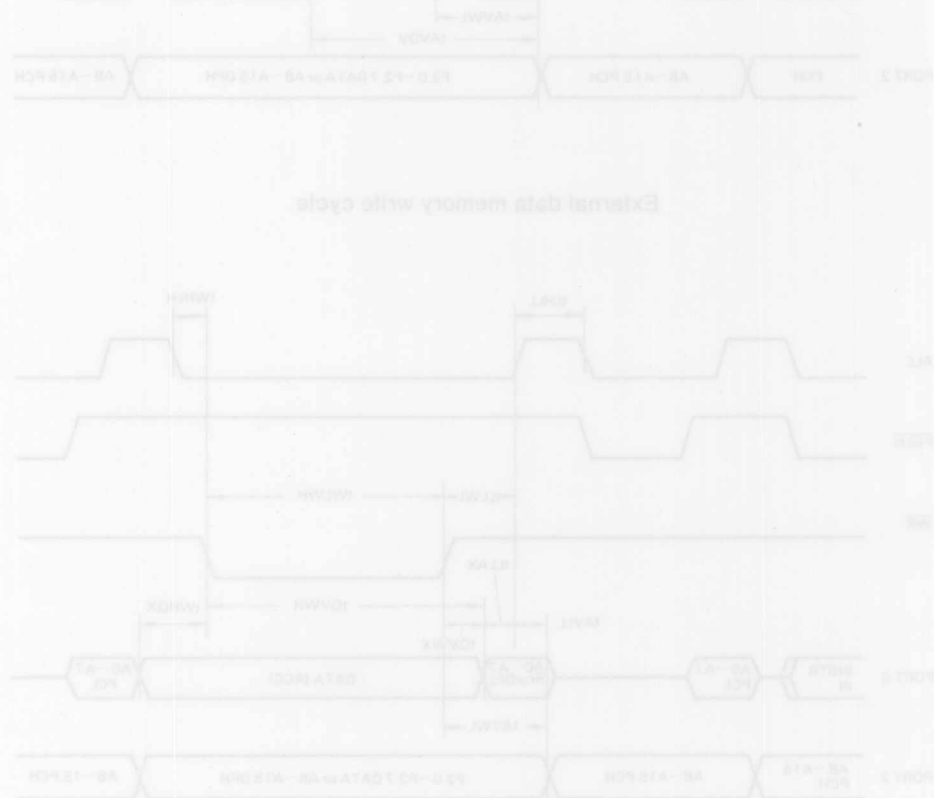
6

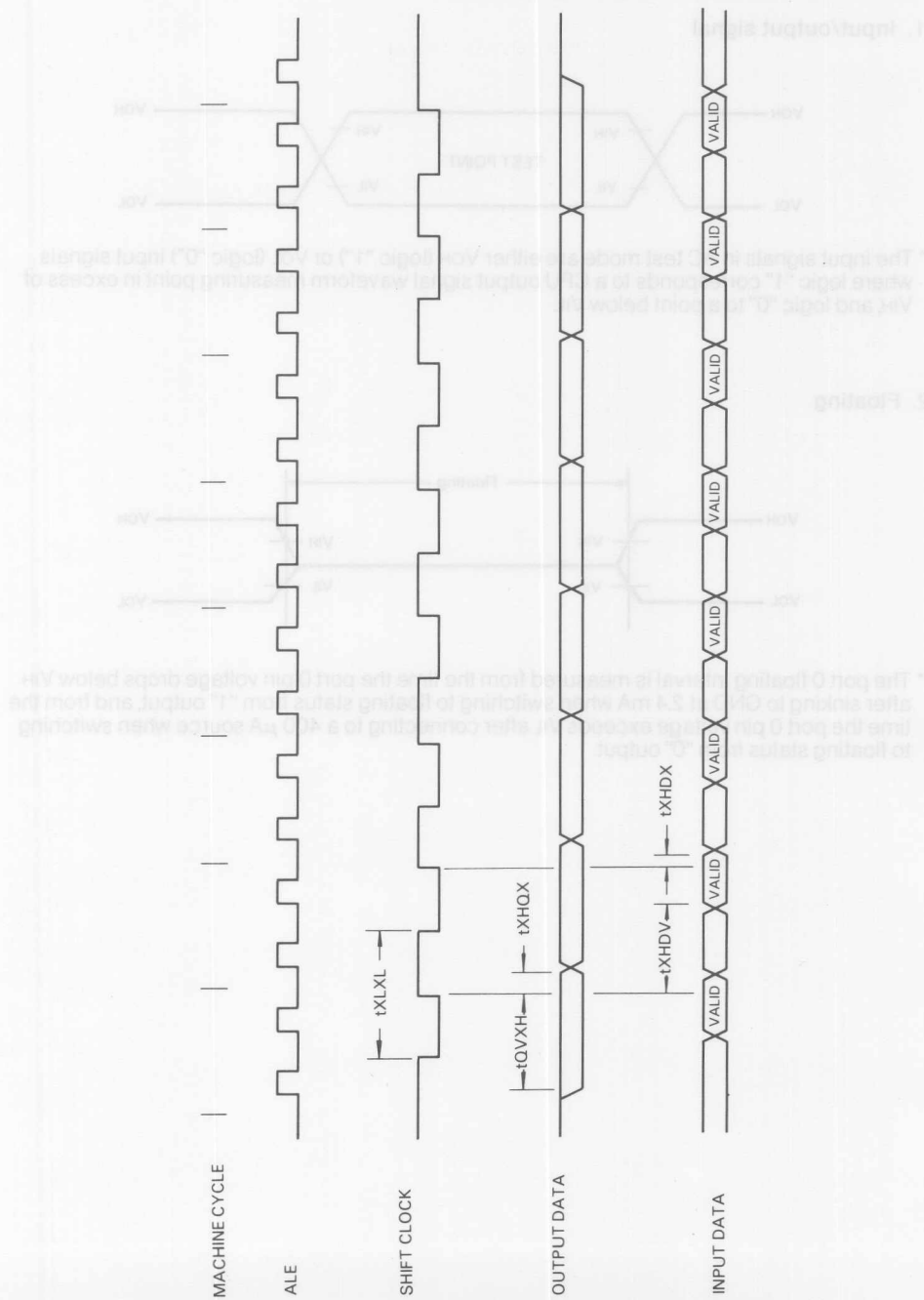


## 6.6 Serial Port (I/O Extension Mode) AC Characteristics

$V_{CC} = 5\text{ V} \pm 20\%$ ,  $V_{SS} = 0\text{ V}$ ,  $T_a = -40^\circ\text{C}$  to  $85^\circ\text{C}$ )

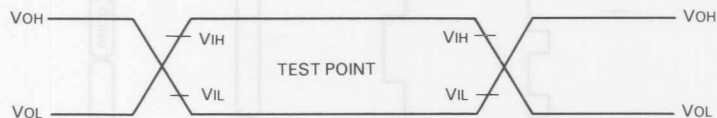
Parameter	Symbol	Min	Max	Unit
Serial Port Clock Cycle Time	tXLXL	12tCLCL		$\mu\text{S}$
Output Data Setup to Clock Rising Edge	tQVXH	10tCLCL-133		nS
Output Data Hold After Clock Rising Edge	tXHQX	2tCLCL-177		nS
Input Data Hold After Clock Rising Edge	tXHDX	0		nS
Clock Rising Edge to Input Data Valid	tXHDV		10tCLCL-133	nS





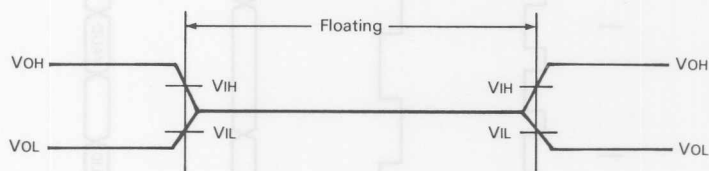
## 6.7 AC Characteristics Measuring Conditions

### 1. Input/output signal



\* The input signals in AC test mode are either  $V_{OH}$  (logic "1") or  $V_{OL}$  (logic "0") input signals where logic "1" corresponds to a CPU output signal waveform measuring point in excess of  $V_{IH}$ , and logic "0" to a point below  $V_{IL}$ .

### 2. Floating



\* The port 0 floating interval is measured from the time the port 0 pin voltage drops below  $V_{IH}$  after sinking to GND at 2.4 mA when switching to floating status from "1" output, and from the time the port 0 pin voltage exceeds  $V_{IL}$  after connecting to a 400  $\mu$ A source when switching to floating status from "0" output.

## 7. DESCRIPTION OF INSTRUCTIONS

# 7. DESCRIPTION OF INSTRUCTIONS

- (15) Other instruction (1)
- (14) External data memory instructions (4)
- (13) Branching instructions (13)
- (12) Jump instructions (4)
- (11) Subroutine instructions (6)
- (10) Data exchange instructions (4)
- (9) Constant value instructions (2)
- (8) Data transfer instructions (17)
- (7) Bit operation instructions (2)
- (6) Carry flag operation instructions (6)
- (5) Immediate data setting instructions (2)
- (4) Logical operation instructions (18)

## 7. DESCRIPTION OF INSTRUCTIONS

### 7.1 Outline

MSM80C31/MSM80C51 is a microcontroller designed for parallel processing in an 8-bit ALU. The instructions consist of 8-bit units of data, and are available as 1-word 1-machine cycle, 2-machine cycle, and 4-machine cycle instructions as well as 2-word 1-machine cycle and 2-machine cycle instructions and 3-word 2-machine cycle instructions. There is a total of 111 instructions classified into the following groups.

- (1) Arithmetic and logic instructions (15)
- (2) Accumulator operation instructions (7)
- (3) Increment & decrement instructions (9)
- (4) Logical operation instructions (18)
- (5) Immediate data setting instructions (5)
- (6) Carry flag operation instructions (9)
- (7) Bit operation instructions (3)
- (8) Data transfer instructions (11)
- (9) Constant value instructions (2)
- (10) Data exchange instructions (4)
- (11) Subroutine instructions (6)
- (12) Jump instructions (4)
- (13) Branching instructions (13)
- (14) External data memory instructions (4)
- (15) Other instruction (1)

## 7.2 Description of Instruction Symbols

A description of the instruction symbols is listed below:

A	Accumulator
AB	Register pair
AC	Auxiliary carry
B	Arithmetic operation register
C	Carry (the bit 7 carry represented by CY is changed to C in Chapter 7.)
DPTR	Data pointer
PC	Program counter
Rr	Register representation (r = 0/1, or r = 0 thru 7)
SP	Stack pointer
AND	Logical AND
OR	Logical OR
XOR	Exclusive OR
+	Addition
-	Subtraction
×	Multiplication
/	Division
(X)	Representation of the contents of X
((X))	Representation of the contents addressed by contents of X
#	Symbol denoting immediate data
@	Symbol denoting indirect address
=	Equal
≠	Not equal
←	Substitution
→	Substitution
—	Negation (upper bar)
<	Smaller than
>	Larger than
bit address	RAM or special function register bit designated address
code address	Absolute address (A <sub>0</sub> thru A <sub>15</sub> , A <sub>0</sub> thru A <sub>11</sub> )
data	Immediate data (I <sub>0</sub> thru I <sub>7</sub> )
relative offset	Corrected relative jump address value
direct address	RAM or special function register data designated address ("direct address" representation changed to "data address" during detailed description of instructions)

## 7.3 List of Instructions

MSM80C31/MSM80C51 instruction table

L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
H	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
0 0000	NOP	AJMP address 11 (page 0)	LJMP address 16	RR A	INC A	INC direct	INC @R0	INC @R1	INC R0	INC R1	INC R2	INC R3	INC R4	INC R5	INC R6	INC R7
1 0001	JBC bit, rel	ACALL address 11 (page 0)	LCALL address 16	RRC A	DEC A	DEC direct	DEC @R0	DEC @R1	DEC R0	DEC R1	DEC R2	DEC R3	DEC R4	DEC R5	DEC R6	DEC R7
2 0010	JB bit, rel	AJMP address 11 (page 1)	RET	RL A	ADD A, #data	ADD A, direct	ADDA, @R0	ADDA, @R1	ADD A, R0	ADD A, R1	ADD A, R2	ADD A, R3	ADD A, R4	ADD A, R5	ADD A, R6	ADD A, R7
3 0011	JNB bit, rel	ACALL address 11 (page 1)	RET1	RLCA	ADDC A, #data	ADDC A, direct	ADDC A, @R0	ADDC A, @R1	ADDC A, R0	ADDC A, R1	ADDC A, R2	ADDC A, R3	ADDC A, R4	ADDC A, R5	ADDC A, R6	ADDC A, R7
4 0100	JC bit, rel	AJMP address 11 (page 2)	ORL direct, A	ORL direct, #data	ORL A, #data	ORL A, direct	ORL A, @R0	ORL A, @R1	ORLA, R0	ORLA, R1	ORLA, R2	ORLA, R3	ORLA, R4	ORLA, R5	ORLA, R6	ORLA, R7
5 0101	JNC rel	ACALL address 11 (page 2)	ANL direct, A	ANL direct, #data	ANL A, #data	ANL A, direct	ANL A, @R0	ANL A, @R1	ANLA, R0	ANLA, R1	ANLA, R2	ANLA, R3	ANLA, R4	ANLA, R5	ANLA, R6	ANLA, R7
6 0110	JZ rel	AJMP address 11 (page 3)	XRL direct, A	XRL direct, #data	XRL A, #data	XRL A, direct	XRL A, @R0	XRL A, @R1	XRLA, R0	XRLA, R1	XRLA, R2	XRLA, R3	XRLA, R4	XRLA, R5	XRLA, R6	XRLA, R7
7 0111	JNZ rel	ACALL address 11 (page 3)	ORL C, bit	JMP @A+DPTR	MOVA, #data	MOV direct, #data	MOV @R0, #data	MOV @R1, #data	MOV R0, #data	MOV R1, #data	MOV R2, #data	MOV R3, #data	MOV R4, #data	MOV R5, #data	MOV R6, #data	MOV R7, #data
8 1000	SJMP rel	AJMP address 11 (page 4)	ANL C, bit	MOVC A, @A+PC	DIVAB	MOV direct 1, direct 2	MOV direct, @R0	MOV direct, @R1	MOV direct, R0	MOV direct, R1	MOV direct, R2	MOV direct, R3	MOV direct, R4	MOV direct, R5	MOV direct, R6	MOV direct, R7
9 1001	MOV DPTR, #data 16	ACALL address 11 (page 4)	MOV bit, C	MOVC A, @A+DPTR	SUBB A, #data	SUBB A, direct	SUBB A, @R0	SUBB A, @R1	SUBB A, R0	SUBB A, R1	SUBB A, R2	SUBB A, R3	SUBB A, R4	SUBB A, R5	SUBB A, R6	SUBB A, R7
A 1010	ORL C,/bit	AJMP address 11 (page 5)	MOV C, bit	INC DPTR	MUL AB		MOV @R0, direct	MOV @R1, direct	MOV R0, direct	MOV R1, direct	MOV R2, direct	MOV R3, direct	MOV R4, direct	MOV R5, direct	MOV R6, direct	MOV R7, direct
B 1011	ANL C,/bit	ACALL address 11 (page 5)	CPL bit	CPL C	CJNE A, #data,rel	CJNE A, direct,rel	CJNE @R0, direct,rel	CJNE @R1, direct,rel	CJNE R0, #data,rel	CJNE R1, #data,rel	CJNE R2, #data,rel	CJNE R3, #data,rel	CJNE R4, #data,rel	CJNE R5, #data,rel	CJNE R6, #data,rel	CJNE R7, #data,rel
C 1100	PUSH direct	AJMP address 11 (page 6)	CLR bit	CLRC	SWAP A	XCH A, direct	XCH A, @R0	XCH A, @R1	XCHA, R0	XCHA, R1	XCHA, R2	XCHA, R3	XCHA, R4	XCHA, R5	XCHA, R6	XCHA, R7
D 1101	POP direct	ACALL address 11 (page 6)	SETB bit	SETB C	DA A	DJNZ direct, rel	XCHD A, @R0	XCHD A, @R1	DJNZ R0, rel	DJNZ R1, rel	DJNZ R2, rel	DJNZ R3, rel	DJNZ R4, rel	DJNZ R5, rel	DJNZ R6, rel	DJNZ R7, rel
E 1110	MOVX A, @DPTR	AJMP address 11 (page 7)	MOVX A, @R0	MOVX A, @R1	CLR A	MOVA, direct	MOVA, @R0	MOVA, @R1	MOVA, R0	MOVA, R1	MOVA, R2	MOVA, R3	MOVA, R4	MOVA, R5	MOVA, R6	MOVA, R7
F 1111	MOVX @DPTR,A	ACALL address 11 (page 7)	MOVX @R0, A	MOVX @R1, A	CPL A	MOV direct, A	MOV @R0, A	MOV @R1, A	MOVR0,A	MOVR1,A	MOVR2,A	MOVR3,A	MOVR4,A	MOVR5,A	MOVR6,A	MOVR7,A

## 7.4 Simplified Description of Instructions

Note that "data address" is represented as "direct address" in this description.

Classification	Mnemonic	Instruction code								Byte	Cycle	Description	Page
		D7	D6	D5	D4	D3	D2	D1	D0				
Arithmetic operation instructions	ADD A, Rr	0	0	1	0	1	r <sub>2</sub>	r <sub>1</sub>	r <sub>0</sub>	1	1	(AC),(OV),(C),(A) ← (A)+(Rr) r=0~7	177
	ADD A, direct	0	0	1	0	0	1	0	1	2	1	(AC),(OV),(C),(A) ← (A)+(direct address)	178
	ADD A, @Rr	0	0	1	0	0	1	1	r <sub>0</sub>	1	1	(AC),(OV),(C),(A) ← (A)+((Rr)) r=0 or 1	176
	ADD A, #data	0	0	1	0	0	1	0	0	2	1	(AC),(OV),(C),(A) ← (A)+#data	175
	ADDC A, Rr	0	0	1	1	1	r <sub>2</sub>	r <sub>1</sub>	r <sub>0</sub>	1	1	(AC),(OV),(C),(A) ← (A)+(C)+(Rr) r=0~7	181
	ADDC A, direct	0	0	1	1	0	1	0	1	2	1	(AC),(OV),(C),(A) ← (A)+(C)+(direct address)	182
	ADDC A, @Rr	0	0	1	1	0	1	1	r <sub>0</sub>	1	1	(AC),(OV),(C),(A) ← (A)+(C)+((Rr)) r=0 or 1	180
	ADDC A, #data	0	0	1	1	0	1	0	0	2	1	(AC),(OV),(C),(A) ← (A)+(C)+#data	179
	SUBB A, Rr	1	0	0	1	1	r <sub>2</sub>	r <sub>1</sub>	r <sub>0</sub>	1	1	(AC),(OV),(C),(A) ← (A)-((C)+(Rr)) r=0~7	287
	SUBB A, direct	1	0	0	1	0	1	0	1	2	1	(AC),(OV),(C),(A) ← (A)-((C)+(direct address))	288
	SUBB A, @Rr	1	0	0	1	0	1	1	r <sub>0</sub>	1	1	(AC),(OV),(C),(A) ← (A)-((C)+((Rr))) r=0 or 1	286
	SUBB A, #data	1	0	0	1	0	1	0	0	2	1	(AC),(OV),(C),(A) ← (A)-((C)+#data)	285
	MUL AB	1	0	1	0	0	1	0	0	1	4	(AB) ← (A) × (B)	263
	DIV AB	1	0	0	0	0	1	0	0	1	4	(A) quotient, (B) remainder ← (A)/(B)	212
	DA A	1	1	0	1	0	1	0	0	1	1	If the contents of accumulator bits 0 thru 3 exceed 9, or the auxiliary carry (AC) is 1, 6 is added to bits 0 thru 3. And if examination of bits 4 thru 7 shows that the result of adding the carry following correction of the lower order bits 0 thru 3 by 6 is in excess of 9, or carry (C) is 1, 6 is added to bits 4 thru 7. If a carry is generated as a result, 1 is set in the carry flag.	206



Classification	Mnemonic	Instruction code								Byte	Cycle	Description	Page
		D7	D6	D5	D4	D3	D2	D1	D0				
Accumulator operation instructions	CLR A	1	1	1	0	0	1	0	0	1	1	$(A) \leftarrow 0$	200
	CPL A	1	1	1	1	0	1	0	0	1	1	$(A) \leftarrow \overline{(A)}$	203
	RL A	0	0	1	0	0	0	1	1	1	1		277
	RLC A	0	0	1	1	0	0	1	1	1	1		278
	RR A	0	0	0	0	0	0	1	1	1	1		279
	RRC A	0	0	0	1	0	0	1	1	1	1		280
	SWAP A	1	1	0	0	0	1	0	0	1	1	$(A4 \sim 7) \rightleftharpoons (A0 \sim 3)$	289

Classification	Mnemonic		Instruction code								Byte	Cycle	Description	Page
			D7	D6	D5	D4	D3	D2	D1	D0				
Increment & decrement instructions	INC	A	0	0	0	0	0	1	0	0	1	1	(A) ← (A)+1	218
	INC	Rr	0	0	0	0	1	r <sub>2</sub>	r <sub>1</sub>	r <sub>0</sub>	1	1	(Rr) ← (Rr)+1 r=0~7	220
	INC	direct	0	0	0	0	0	1	0	1	2	1	(direct address) ← (direct address) + 1	221
			a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>				
	INC	@Rr	0	0	0	0	0	1	1	r <sub>0</sub>	1	1	((Rr)) ← ((Rr)) + 1 r=0 or 1	217
	INC	DPTR	1	0	1	0	0	0	1	1	1	2	(DPTR) ← (DPTR) + 1	219
	DEC	A	0	0	0	1	0	1	0	0	1	1	(A) ← (A) - 1	209
	DEC	Rr	0	0	0	1	1	r <sub>2</sub>	r <sub>1</sub>	r <sub>0</sub>	1	1	(Rr) ← (Rr) - 1 r=0~7	210
	DEC	direct	0	0	0	1	0	1	0	1	2	1	(direct address) → (direct address) - 1	211
		a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>					
	DEC	@Rr	0	0	0	1	0	1	1	r <sub>0</sub>	1	1	((Rr)) ← ((Rr)) - 1 r=0 or 1	208
Logical operation instructions	ANL	A, Rr	0	1	0	1	1	r <sub>2</sub>	r <sub>1</sub>	r <sub>0</sub>	1	1	(A) ← (A) AND (Rr) r=0~7	186
	ANL	A, direct	0	1	0	1	0	1	0	1	2	1	(A) ← (A) AND (direct address)	187
			a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>				
	ANL	A, @Rr	0	1	0	1	0	1	1	r <sub>0</sub>	1	1	(A) ← (A) AND ((Rr)) r=0 or 1	185
	ANL	A, #data	0	1	0	1	0	1	0	0	2	1	(A) ← (A) AND #data	184
			l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>				
	ANL	direct, A	0	1	0	1	0	0	1	0	2	1	(direct address) ← (direct address) AND (A)	191
			a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>				
	ANL	direct, #data	0	1	0	1	0	0	1	1	3	2	(direct address) ← (direct address) AND #data	190
			a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>				
		l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>					
	ORL	A, Rr	0	1	0	0	1	r <sub>2</sub>	r <sub>1</sub>	r <sub>0</sub>	1	1	(A) ← (A) OR (Rr) r=0~7	267
	ORL	A, direct	0	1	0	0	0	1	0	1	2	1	(A) ← (A) OR (direct address)	268
		a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>					
	ORL	A, @Rr	0	1	0	0	0	1	1	r <sub>0</sub>	1	1	(A) ← (A) OR ((Rr)) r=0 or 1	266
	ORL	A, #data	0	1	0	0	0	1	0	0	2	1	(A) ← (A) OR #data	265
		l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>					

Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
Logical operation instructions	ORL direct, A	0 1 0 0 0 0 1 0 a7 a6 a5 a4 a3 a2 a1 a0	2	1	(direct address) ← (direct address) OR (A)	272
	ORL direct, #data	0 1 0 0 0 0 1 1 a7 a6 a5 a4 a3 a2 a1 a0 l7 l6 l5 l4 l3 l2 l1 l0	3	2	(direct address) ← (direct address) OR #data	271
	XRL A, Rr	0 1 1 0 1 r2 r1 r0	1	1	(A) ← (A) XOR (Rr) r=0~7	296
	XRL A, direct	0 1 1 0 0 1 0 1 a7 a6 a5 a4 a3 a2 a1 a0	2	1	(A) ← (A) XOR (direct address)	297
	XRL A, @Rr	0 1 1 0 0 1 1 r0	1	1	(A) ← (A) XOR ((Rr)) r=0 or 1	295
	XRL A, #data	0 1 1 0 0 1 0 0 l7 l6 l5 l4 l3 l2 l1 l0	2	1	(A) ← (A) XOR #data	294
	XRL direct, A	0 1 1 0 0 0 1 0 a7 a6 a5 a4 a3 a2 a1 a0	2	1	(direct address) ← (direct address) XOR (A)	299
	XRL direct, #data	0 1 1 0 0 0 1 1 a7 a6 a5 a4 a3 a2 a1 a0 l7 l6 l5 l4 l3 l2 l1 l0	3	2	(direct address) ← (direct address) XOR #data	298
Immediate data setting instructions	MOV A, #data	0 1 1 1 0 1 0 0 l7 l6 l5 l4 l3 l2 l1 l0	2	1	(A) ← #data	242
	MOV Rr, #data	0 1 1 1 1 r2 r1 r0 l7 l6 l5 l4 l3 l2 l1 l0	2	1	(Rr) ← #data r=0~7	248
	MOV direct, #data	0 1 1 1 0 1 0 1 a7 a6 a5 a4 a3 a2 a1 a0 l7 l6 l5 l4 l3 l2 l1 l0	3	2	(direct address) ← #data	252
	MOV @Rr, #data	0 1 1 1 0 1 1 r0 l7 l6 l5 l4 l3 l2 l1 l0	2	1	((Rr)) ← #data r=0 or 1	239
	MOV DPTR, #data 16	1 0 0 1 0 0 0 0 l15 l14 l13 l12 l11 l10 l9 l8 l7 l6 l5 l4 l3 l2 l1 l0	3	2	(DPTR) ← #data 16	247

Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
Carry flag operation instructions	CLR C	1 1 0 0 0 0 1 1	1	1	$(C) \leftarrow 0$	201
	SETB C	1 1 0 1 0 0 1 1	1	1	$(C) \leftarrow 1$	281
	CPL C	1 0 1 1 0 0 1 1	1	1	$(C) \leftarrow \overline{(C)}$	204
	ANL C, bit	1 0 0 0 0 0 1 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	2	$(C) \leftarrow (C) \text{ AND } (\text{bit address})$	188
	ANL C, /bit	1 0 1 1 0 0 0 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	2	$(C) \leftarrow (C) \text{ AND } \overline{(\text{bit address})}$	189
	ORL C, bit	0 1 1 1 0 0 1 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	2	$(C) \leftarrow (C) \text{ OR } (\text{bit address})$	269
	ORL C, /bit	1 0 1 0 0 0 0 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	2	$(C) \leftarrow (C) \text{ OR } \overline{(\text{bit address})}$	270
	MOV C, bit	1 0 1 0 0 0 1 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	1	$(C) \leftarrow (\text{bit address})$	246
	MOV bit, C	1 0 0 1 0 0 1 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	2	$(\text{bit address}) \leftarrow (C)$	251
Bit operation instructions	SETB bit	1 1 0 1 0 0 1 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	1	$(\text{bit address}) \leftarrow 1$	282
	CLR bit	1 1 0 0 0 0 1 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	1	$(\text{bit address}) \leftarrow 0$	202
	CPL bit	1 0 1 1 0 0 1 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	2	1	$(\text{bit address}) \leftarrow \overline{(\text{bit address})}$	205

Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
Data transfer instructions	MOV A, Rr	1 1 1 0 1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	1	1	(A) ← (Rr) r=0~7	244
	MOV A, direct	1 1 1 0 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	1	(A) ← (direct address)	245
	MOV A, @Rr	1 1 1 0 0 1 1 r <sub>0</sub>	1	1	(A) ← ((Rr)) r=0 or 1	243
	MOV Rr, A	1 1 1 1 1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	1	1	(Rr) ← (A) r=0~7	249
	MOV Rr, direct	1 0 1 0 1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub> a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	2	(Rr) ← (direct address) r=0~7	250
	MOV direct, A	1 1 1 1 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	1	(direct address) ← (A)	254
	MOV direct, Rr	1 0 0 0 1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub> a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	2	(direct address) ← (Rr) r=0~7	255
	MOV direct 1, direct 2	1 0 0 0 0 1 0 1 a <sub>7</sub> <sup>2</sup> a <sub>6</sub> <sup>2</sup> a <sub>5</sub> <sup>2</sup> a <sub>4</sub> <sup>2</sup> a <sub>3</sub> <sup>2</sup> a <sub>2</sub> <sup>2</sup> a <sub>1</sub> <sup>2</sup> a <sub>0</sub> <sup>2</sup> a <sub>7</sub> <sup>1</sup> a <sub>6</sub> <sup>1</sup> a <sub>5</sub> <sup>1</sup> a <sub>4</sub> <sup>1</sup> a <sub>3</sub> <sup>1</sup> a <sub>2</sub> <sup>1</sup> a <sub>1</sub> <sup>1</sup> a <sub>0</sub> <sup>1</sup>	3	2	(direct address 1) ← (direct address 2)	256
	MOV direct, @Rr	1 0 0 0 0 1 1 r <sub>0</sub> a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	2	(direct address) ← ((Rr)) r=0 or 1	253
	MOV @Rr, A	1 1 1 1 0 1 1 r <sub>0</sub>	1	1	((Rr)) ← (A) r=0 or 1	240
	MOV @Rr, direct	1 0 1 0 0 1 1 r <sub>0</sub> a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	2	((Rr)) ← (direct address) r=0 or 1	241
Constant value instructions	MOVC A, @A+DPTR	1 0 0 1 0 0 1 1	1	2	(A) ← ((A) + (DPTR))	257
	MOVC A, @A+PC	1 0 0 0 0 0 1 1	1	2	(PC) ← (PC) + 1 (A) ← ((A) + (PC))	258

Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
Data exchange instructions	XCH A, Rr	1 1 0 0 1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	1	1	(A) $\rightleftharpoons$ (Rr) r=0~7	291
	XCH A, direct	1 1 0 0 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	1	(A) $\rightleftharpoons$ (direct address)	292
	XCH A, @Rr	1 1 0 0 0 1 1 r <sub>0</sub>	1	1	(A) $\rightleftharpoons$ ((Rr)) r=0 or 1	290
	XCHD A, @Rr	1 1 0 1 0 1 1 r <sub>0</sub>	1	1	(A <sub>0-3</sub> ) $\rightleftharpoons$ ((Rr <sub>0-3</sub> )) r=0 or 1	293
Subroutine instructions	PUSH direct	1 1 0 0 0 0 0 0 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	2	(SP) $\leftarrow$ (SP) + 1 ((SP)) $\leftarrow$ (direct address)	274
	POP direct	1 1 0 1 0 0 0 0 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	2	2	(direct address) $\leftarrow$ ((SP)) (SP) $\leftarrow$ (SP) - 1	273
	ACALL addr 11	A <sub>10</sub> A <sub>9</sub> A <sub>8</sub> 1 0 0 0 1 A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub>	2	2	(PC) $\leftarrow$ (PC) + 2 (SP) $\leftarrow$ (SP) + 1 ((SP)) $\leftarrow$ (PC <sub>0-7</sub> ) (SP) $\leftarrow$ (SP) + 1 ((SP)) $\leftarrow$ (PC <sub>8-15</sub> ) (PC <sub>0-10</sub> ) $\leftarrow$ A <sub>0-10</sub>	174
	LCALL addr 16	0 0 0 1 0 0 1 0 A <sub>15</sub> A <sub>14</sub> A <sub>13</sub> A <sub>12</sub> A <sub>11</sub> A <sub>10</sub> A <sub>9</sub> A <sub>8</sub> A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub>	3	2	(PC) $\leftarrow$ (PC) + 3 (SP) $\leftarrow$ (SP) + 1 ((SP)) $\leftarrow$ (PC <sub>0-7</sub> ) (SP) $\leftarrow$ (SP) + 1 ((SP)) $\leftarrow$ (PC <sub>8-15</sub> ) (PC <sub>0-15</sub> ) $\leftarrow$ A <sub>0-15</sub>	237

Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
Subroutine instructions	RET	0 0 1 0 0 0 1 0	1	2	(PC <sub>8-15</sub> ) ← ((SP)) (SP) ← (SP) - 1 (PC <sub>0-7</sub> ) ← ((SP)) (SP) ← (SP) - 1	275
	RETI	0 0 1 1 0 0 1 0	1	2	(PC <sub>8-15</sub> ) ← ((SP)) (SP) ← (SP) - 1 (PC <sub>0-7</sub> ) ← ((SP)) (SP) ← (SP) - 1 * INTERRUPT ENABLE	276
Jump instructions	AJMP addr 11	A <sub>10</sub> A <sub>9</sub> A <sub>8</sub> 0 0 0 0 1 A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub>	2	2	(PC) ← (PC) + 2 (PC <sub>0-10</sub> ) ← A <sub>0-10</sub>	183
	LJMP addr 16	0 0 0 0 0 0 1 0 A <sub>15</sub> A <sub>14</sub> A <sub>13</sub> A <sub>12</sub> A <sub>11</sub> A <sub>10</sub> A <sub>9</sub> A <sub>8</sub> A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub>	3	2	(PC <sub>0-15</sub> ) ← A <sub>0-15</sub>	238
	SJMP rel	1 0 0 0 0 0 0 0 R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	2	2	(PC) ← (PC) + 2 (PC) ← (PC) + relative offset	283
	JMP @A+DPTR	0 1 1 1 0 0 1 1	1	2	(PC) ← (A) + (DPTR)	228

Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
Branching instructions	CJNE A, direct, rel	1 0 1 1 0 1 0 1 a7 a6 a5 a4 a3 a2 a1 a0 R7 R6 R5 R4 R3 R2 R1 R0	3	2	(PC) ← (PC) + 3 IF (A) ≠ (direct address) THEN (PC) ← (PC) + relative offset IF (A) < (direct address) THEN (C) ← 1 ELSE (C) ← 0	196
	CJNE A, #data, rel	1 0 1 1 0 1 0 0 l7 l6 l5 l4 l3 l2 l1 l0 R7 R6 R5 R4 R3 R2 R1 R0	3	2	(PC) ← (PC) + 3 IF (A) ≠ #data THEN (PC) ← (PC) + relative offset IF (A) < #data THEN (C) ← 1 ELSE (C) ← 0	194
	CJNE Rr, #data, rel	1 0 1 1 1 r2 r1 r0 l7 l6 l5 l4 l3 l2 l1 l0 R7 R6 R5 R4 R3 R2 R1 R0	3	2	(PC) ← (PC) + 3 IF (Rr) ≠ #data r=0~7 THEN (PC) ← (PC) + relative offset IF (Rr) < #data r=0~7 THEN (C) ← 1 ELSE (C) ← 0	198



Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
Branching instructions	CJNE @Rr, #data, rel	1 0 1 1 0 1 1 r <sub>0</sub> l <sub>7</sub> l <sub>6</sub> l <sub>5</sub> l <sub>4</sub> l <sub>3</sub> l <sub>2</sub> l <sub>1</sub> l <sub>0</sub> R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	3	2	(PC) ← (PC) + 3 IF ((Rr) ≠ #data r=0 or 1 THEN (PC) ← (PC) + relative offset IF ((Rr) < #data r=0 or 1 THEN (C) ← 1 ELSE (C) ← 0	192
	DJNZ Rr, rel	1 1 0 1 1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub> R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	2	2	(PC) ← (PC) + 2 (Rr) ← (Rr) - 1 r=0~7 IF (Rr) ≠ 0 r=0~7 THEN (PC) ← (PC) + relative offset	213
	DJNZ direct, rel	1 1 0 1 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub> R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	3	2	(PC) ← (PC) + 3 (direct address) ← (direct address) - 1 IF (direct address) ≠ 0 THEN (PC) ← (PC) + relative offset	215

Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
Branching instructions	JZ rel	0 1 1 0 0 0 0 0 R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	2	2	(PC) ← (PC) + 2 IF (A) = 0 THEN (PC) ← (PC) + relative offset	235
	JNZ rel	0 1 1 1 0 0 0 0 R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	2	2	(PC) ← (PC) + 2 IF (A) ≠ 0 THEN (PC) ← (PC) + relative offset	233
	JC rel	0 1 0 0 0 0 0 0 R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	2	2	(PC) ← (PC) + 2 IF (C) = 1 THEN (PC) ← (PC) + relative offset	226
	JNC rel	0 1 0 1 0 0 0 0 R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	2	2	(PC) ← (PC) + 2 IF (C) = 0 THEN (PC) ← (PC) + relative offset	231
	JB bit, rel	0 0 1 0 0 0 0 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	3	2	(PC) ← (PC) + 3 IF (bit address) = 1 THEN (PC) ← (PC) + relative offset	222
	JNB bit, rel	0 0 1 1 0 0 0 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	3	2	(PC) ← (PC) + 3 IF (bit address) = 0 THEN (PC) ← (PC) + relative offset	229
	JBC bit, rel	0 0 0 1 0 0 0 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> R <sub>7</sub> R <sub>6</sub> R <sub>5</sub> R <sub>4</sub> R <sub>3</sub> R <sub>2</sub> R <sub>1</sub> R <sub>0</sub>	3	2	(PC) ← (PC) + 3 IF (bit address) = 1 THEN (bit address) ← 0 (PC) ← (PC) + relative offset	224

Classification	Mnemonic	Instruction code	Byte	Cycle	Description	Page
		D7 D6 D5 D4 D3 D2 D1 D0				
External memory instructions	MOVX A, @Rr	1 1 1 0 0 0 1 r <sub>0</sub>	1	2	(A) ← ((Rr))      EXTERNAL RAM      r=0 or 1	262
	MOVX A, @DPTR	1 1 1 0 0 0 0 0	1	2	(A) ← ((DPTR))      EXTERNAL RAM	261
	MOVX @Rr, A	1 1 1 1 0 0 1 r <sub>0</sub>	1	2	((Rr)) ← (A)      EXTERNAL RAM      r=0 or 1	260
	MOVX @DPTR, A	1 1 1 1 0 0 0 0	1	2	((DPTR)) ← (A)      EXTERNAL RAM	259
Other instruction	NOP	0 0 0 0 0 0 0 0	1	1	(PC) ← (PC) + 1	264

## 7.5 Detailed Description of MSM80C31/MSM80C51 Instructions

Note: "direct address" is represented as "data address" in this detailed description.

Instruction code	Call address	Operation	Number of bytes	Number of cycles	Flags	(PSW)	Description
Byte 1 A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub>	Byte 2 A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub>	$(PC) \leftarrow (PC) + 2$ $(SP) \leftarrow (SP) + 1$ $(SP) \leftarrow (PC) - 1$ $(SP) \leftarrow (SP) + 1$ $(SP) \leftarrow (PC) - 1$ $(PC) \leftarrow A_{16}$	2	2	C AC FO RS1 RS0 OV FI P		<p>This instruction stores the program counter value (current address) in the stack following an increment operation.</p> <p>The program counter data <math>PC \leftarrow PC + 2</math> following <math>PC + 2</math> is replaced by 17-bit page address data <math>A_{16}</math>. The destination address for this instruction must always be within the 2K byte page, but if the instruction is placed at address X7FEH or X7FFH, execution proceeds from the call address on the next page.</p>

## 1. ACALL code address (Absolute call within 2K byte page)

Instruction code	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>A<sub>10</sub></td><td>A<sub>9</sub></td><td>A<sub>8</sub></td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	7							0	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	1	0	0	0	1	Byte 1
7							0												
A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	1	0	0	0	1												
Call address	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>A<sub>7</sub></td><td>A<sub>6</sub></td><td>A<sub>5</sub></td><td>A<sub>4</sub></td><td>A<sub>3</sub></td><td>A<sub>2</sub></td><td>A<sub>1</sub></td><td>A<sub>0</sub></td></tr></table>	7							0	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	Byte 2
7							0												
A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>												
Operation	:	$(PC) \leftarrow (PC) + 2$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{0-7})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{8-15})$ $(PC_{0-10}) \leftarrow A_{0-10}$																	
Number of bytes	:	2																	
Number of cycles	:	2																	
Flags (PSW)	:	<table><tr><td>C</td><td>AC</td><td>F0</td><td>RS1</td><td>RS0</td><td>OV</td><td>F1</td><td>P</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>	C	AC	F0	RS1	RS0	OV	F1	P									
C	AC	F0	RS1	RS0	OV	F1	P												
Description	:	<p>This instruction stores the program counter value (return address) in the stack following an increment operation.</p> <p>The program counter data PC<sub>0</sub> ~ PC<sub>10</sub> following PC+2 is replaced by 11-bit page address data A<sub>0</sub> ~ A<sub>10</sub>. The destination address for this instruction must always be within the 2K byte page, but if the instruction is placed at address X7FEH or X7FFH, execution proceeds from the call address on the next page.</p>																	

2. ADD A, #data (Add immediate data)

Instruction code	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	0	0	1	0	0	1	0	0	Byte 1
7	6	5	4	3	2	1	0												
0	0	1	0	0	1	0	0												
# data	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td></tr></table>	7	6	5	4	3	2	1	0	17	16	15	14	13	12	11	10	Byte 2
7	6	5	4	3	2	1	0												
17	16	15	14	13	12	11	10												
Operation	:	$(A) \leftarrow (A) + \#data$																	
Number of bytes	:	2																	
Number of cycles	:	1																	
Flags (PSW)	:	C	AC	F0	RS1	RS0	OV	F1	P										
		•	•				•		•										
Description	:	An 8-bit immediate data value is added to the accumulator. The result is placed in the accumulator and the flags are updated.																	

Example ADD A, #07H

Instruction code	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	0	0	1	0	0	1	0	0	Byte 1
7	6	5	4	3	2	1	0												
0	0	1	0	0	1	0	0												
	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	7	6	5	4	3	2	1	0	0	0	0	0	0	1	1	1	Byte 2
7	6	5	4	3	2	1	0												
0	0	0	0	0	1	1	1												
Before execution																			
Accumulator																			
	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	0	1	1	0	0	0	1	0	
7	6	5	4	3	2	1	0												
0	1	1	0	0	0	1	0												
After execution																			
Accumulator																			
	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	7	6	5	4	3	2	1	0	0	1	1	0	1	1	0	1	
7	6	5	4	3	2	1	0												
0	1	1	0	1	1	0	1												

**3. ADD A, @Rr (Add indirect address)**

																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

Example    `ADD A, @R0`

Instruction code: 7 0 : 0 0 1 0 0 1 1 0 Byte 1

Before execution	After execution
Accumulator 7 0 : 0 1 0 0 1 1 0 1	Accumulator 7 0 : 1 0 1 1 0 1 1 0
Register 0 7 0 : 0 1 0 1 1 1 0 0	Register 0 7 0 : 0 1 0 1 1 1 0 0
5CH 7 0 : 0 1 1 0 1 0 0 1	5CH 7 0 : 0 1 1 0 1 0 0 1

#### 4. ADD A, Rr (Add register)

Instruction code : 

7	0
0	0
1	0
1	r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>

 Byte 1

Operation :  $(A) \leftarrow (A) + (Rr) \quad r = 0 \sim 7$

Number of bytes : 1

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
•	•				•		•

Description : The register r contents are added to the accumulator. The result is placed in the accumulator and the flags are updated.

Example ADD A, R6

Instruction code : 

7	0
0	0
1	0
1	1
1	1
0	0

 Byte 1

Before execution

Accumulator

0	1	0	1	0	1	0	0
7							0

Register 6

0	1	0	1	1	1	0	1
7							0

After execution

Accumulator

1	0	1	1	0	0	0	1
7							0

Register 6

0	1	0	1	1	1	0	1
7							0



5. ADD A, data address (Add memory)

Instruction code	:	<table><tr><td>7</td><td colspan="7"></td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	7								0	0	0	1	0	0	1	0	1	Byte 1
7								0												
0	0	1	0	0	1	0	1													
Data address	:	<table><tr><td>7</td><td colspan="7"></td><td>0</td></tr><tr><td>a<sub>7</sub></td><td>a<sub>6</sub></td><td>a<sub>5</sub></td><td>a<sub>4</sub></td><td>a<sub>3</sub></td><td>a<sub>2</sub></td><td>a<sub>1</sub></td><td>a<sub>0</sub></td></tr></table>	7								0	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2
7								0												
a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>													
Operation	:	(A) ← (A) + (data address)																		
Number of bytes	:	2																		
Number of cycles	:	1																		
Flags	:	C	AC	F0	RS1	RS0	OV	F1	P											
(PSW)		●	●				●		●											

Description : The specified data address contents are added to the accumulator. The result is placed in the accumulator and the flags are updated.

Example ADD A, P1

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0</div> <div>0</div> <div>1</div> <div>0</div> <div>0</div> <div>1</div> <div>0</div> <div>1</div> </div>	Byte 1
	:	<div> <div>7</div> <div>0</div> <div>1</div> <div>0</div> <div>0</div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> </div>	Byte 2
Before execution			
Accumulator		<div> <div>7</div> <div>0</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div>1</div> <div>0</div> </div>	
Port 1 (90H)		<div> <div>7</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> </div>	
After execution			
Accumulator		<div> <div>7</div> <div>0</div> <div>0</div> <div>0</div> <div>1</div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> </div>	
Port 1 (90H)		<div> <div>7</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> </div>	

6. ADDC A, #data (Add carry plus immediate data to accumulator)

Instruction code : 

7	0
0	0
1	1
0	1
0	0

 Byte 1

#data : 

7	0
I <sub>7</sub>	I <sub>6</sub>
I <sub>5</sub>	I <sub>4</sub>
I <sub>3</sub>	I <sub>2</sub>
I <sub>1</sub>	I <sub>0</sub>

 Byte 2

Operation :  $(A) \leftarrow (A) + (C) + \#data$

Number of bytes : 2

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
•	•				•		•

Description : The carry flag is added to the accumulator, and an 8-bit immediate data is added to that result. The result is placed in the accumulator and the flags are updated.

Example ADDC A, #76H

Instruction code : 

7	0
0	0
1	1
0	1
0	0

 Byte 1

7	0
0	1
1	1
0	1
1	0

 Byte 2

Before execution

After execution

Accumulator

Accumulator

7	0
0	1
0	1
1	1
0	0

7	0
1	1
0	1
0	0
0	0

Carry flag

Carry flag

1
---

0
---

7

7. ADDC A, @Rr (Add carry plus indirect address to accumulator)

Instruction code : 

7	0
0	0
1	1
0	1
1	1
r	r

 Byte 1

Operation :  $(A) \leftarrow (A) + (C) + ((Rr))$   $r = 0 \text{ or } 1$

Number of bytes : 1

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
•	•				•		•

Description : The carry flag is added to the accumulator, and the contents of data memory location addressed by the register r contents are added to that result. The result is placed in the accumulator and the flags are updated.

Example ADDC A, @R0

Instruction code : 

7	0
0	0
1	1
0	1
1	1
0	0

 Byte 1

Before execution

Accumulator

1	1	0	1	0	1	0	1
7							0

Register 0

0	1	1	0	1	0	1	1
7							0

6BH

0	1	1	1	1	0	1	1
7							0

Carry flag

0
---

After execution

Accumulator

0	1	0	1	0	0	0	0
7							0

Register 0

0	1	1	0	1	0	1	1
7							0

6BH

0	1	1	1	1	0	1	1
7							0

Carry flag

1
---

8. ADDC A, Rr (Add carry plus register to accumulator)

Instruction code : 

7	0
0	0
1	1
1	1
r <sub>2</sub>	r <sub>1</sub>
r <sub>0</sub>	r <sub>0</sub>

 Byte 1

Operation :  $(A) \leftarrow (A) + (C) + (Rr) \quad r = 0 \sim 7$

Number of bytes : 1

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
•	•				•		•

Description : The carry flag is added to the accumulator, and the register r contents are added to that result. The result is placed in the accumulator and the flags are updated.

Example ADDC A, R2

Instruction code : 

7	0
0	0
1	1
1	0
1	0
0	1
0	0

 Byte 1

Before execution

After execution

Accumulator

Accumulator

0	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---

1	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---

7 0

7 0

Register 2

Register 2

0	1	1	0	1	1	1	0
---	---	---	---	---	---	---	---

0	1	1	0	1	1	1	0
---	---	---	---	---	---	---	---

7 0

7 0

Carry flag

Carry flag

1
---

0
---

9. **ADDC A, data address (Add carry plus memory to accumulator)**

Instruction code : 

7							0
0	0	1	1	0	1	0	1

 Byte 1

Data address

a7	a6	a5	a4	a3	a2	a1	a0
----	----	----	----	----	----	----	----

Byte 2

Operation	: $(A) \leftarrow (A) + (C) + (\text{data address})$
-----------	--

Number of bytes : 2

Number of cycles : 1

Flags	C	AC	F0	RS1	RS0	OV	F1	P
(PSW)	●	●				●		●

Description	: The carry flag is added to the accumulator, and the specified data address contents are added to that result. The result is placed in the accumulator and the flags are updated.
-------------	--

Example    ADDC A, 45H

Instruction code : 

7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1

 Byte 1

7 0

0	1	0	0	0	1	0	1
---	---	---	---	---	---	---	---

Byte 2

Before execution

## Accumulator

0 0 1 1 0 0 1 1

7 0

45H

0	1	0	1	1	1	1	0
7							0

Carry flag

1

After execution

Accumulator

1 0 0 1 0 0 1 0  
7 0

45H

Carry flag

0

10. AJMP code address (Absolute jump within 2K byte page)

Instruction code : 

7	0
A <sub>10</sub> A <sub>9</sub> A <sub>8</sub> 0	0 0 0 1

 Byte 1

Jump address : 

7	0
A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub>	A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub>

 Byte 2

Operation : (PC) ← (PC) + 2  
(PC<sub>0-10</sub>) ← A<sub>0-10</sub>

Number of bytes : 2

Number of cycles : 2

Flags : 

C	AC	F0	RS1	RS0	OV	F1	P
---	----	----	-----	-----	----	----	---

  
(PSW) 

--	--	--	--	--	--	--	--

Description : After an increment, the program counter PC<sub>0</sub> ~ PC<sub>10</sub> is replaced by 11-bit page address data A<sub>0</sub> ~ A<sub>10</sub>. The destination address for this instruction must always be within the 2K byte page, but if the instruction is placed at address X7FEH or X7FFH, execution proceeds from the jump address on the next page.

11. ANL A, #data (Logical AND immediate data to accumulator)

Instruction code : 

7	0
0	1
0	1
0	0

 Byte 1

# data : 

7	0
1 <sub>7</sub>	1 <sub>6</sub>
1 <sub>5</sub>	1 <sub>4</sub>
1 <sub>3</sub>	1 <sub>2</sub>
1 <sub>1</sub>	1 <sub>0</sub>

 Byte 2

Operation : (A) ← (A) AND #data

Number of bytes : 2

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

C	AC	F0	RS1	RS0	OV	F1	P
							●

Description : The logical AND between an 8-bit immediate data value and the accumulator contents is determined. The result is stored in the accumulator and the flag is also updated.

Example ANL A, #0AH

Instruction code : 

7	0
0	1
0	1
0	0

 Byte 1

7	0
0	0
0	0
1	0
1	0

 Byte 2

Before execution

Accumulator

7	0
1	0
1	1
1	1
0	1

After execution

Accumulator

7	0
0	0
0	0
0	1
0	0

12. ANL A, @Rr (Logical AND indirect address to accumulator)

Instruction code : 

7	0
0	1
0	1
1	0
1	1
r	0

 Byte 1

Operation : (A) ← (A) AND ((Rr)) r = 0 or 1

Number of bytes : 1

Number of cycles : 1

Flags : 

C	AC	F0	RS1	RS0	OV	F1	P
(PSW)							●

Description : The logical AND between the accumulator contents and the data memory location contents addressed by the register r is determined. The result is stored in the accumulator and the flag is also updated.

Example ANL A, @R0

Instruction code : 

7	0
0	1
0	1
0	1
1	1
0	0

 Byte 1

Before execution

After execution

Accumulator

Accumulator

1	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---

1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---

7 0

7 0

Register 0

Register 0

0	1	0	1	1	0	0	0
---	---	---	---	---	---	---	---

0	1	0	1	1	0	0	0
---	---	---	---	---	---	---	---

7 0

7 0

RAM 58H

RAM 58H

1	1	1	1	1	0	1	0
---	---	---	---	---	---	---	---

1	1	1	1	1	0	1	0
---	---	---	---	---	---	---	---

7 0

7 0



### 13. ANL A,Rr (Logical AND register to accumulator)

Instruction code : 

7	0
0	1
0	1
1	r <sub>2</sub>
r <sub>1</sub>	r <sub>0</sub>

 Byte 1

Operation : (A) ← (A) AND (Rr) r = 0 ~ 7

Number of bytes : 1

Number of cycles : 1

Flags : 

P	AC	F0	RS1	RS0	OV	F1	P
•							•

  
(PSW)

Description : The logical AND between the accumulator contents and the register r contents is determined. The result is stored in the accumulator and the flag is also updated.

Example ANL A, R5

Instruction code : 

7	0
0	1
0	1
1	1
0	1

 Byte 1

<p>Before execution</p> <p>Accumulator</p> <table border="1" style="margin: auto;"><tr><td style="text-align: center;">7</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td></tr></table> <p>Register 5</p> <table border="1" style="margin: auto;"><tr><td style="text-align: center;">7</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td></tr></table>	7	0	1	1	0	1	1	1	0	1	1	1	7	0	0	1	0	1	1	0	1	0	1	1	<p>After execution</p> <p>Accumulator</p> <table border="1" style="margin: auto;"><tr><td style="text-align: center;">7</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr></table> <p>Register 5</p> <table border="1" style="margin: auto;"><tr><td style="text-align: center;">7</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td></tr></table>	7	0	0	1	0	1	0	1	0	0	0	0	7	0	0	1	0	1	1	0	1	0	1	1
7	0																																																
1	1																																																
0	1																																																
1	1																																																
0	1																																																
1	1																																																
7	0																																																
0	1																																																
0	1																																																
1	0																																																
1	0																																																
1	1																																																
7	0																																																
0	1																																																
0	1																																																
0	1																																																
0	0																																																
0	0																																																
7	0																																																
0	1																																																
0	1																																																
1	0																																																
1	0																																																
1	1																																																

14. ANL A,data address (Logical AND memory to accumulator)

Instruction code : 

7	0
0	1
0	1
0	1
0	1
0	1
0	1
0	1

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub>
a <sub>5</sub>	a <sub>4</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (A) ← (A) AND (data address)

Number of bytes : 2

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

								●
--	--	--	--	--	--	--	--	---

Description : The logical AND between the accumulator contents and the specified data address contents is determined. The result is stored in the accumulator and the flag is also updated.

Example ANL A, P1

Instruction code : 

7	0
0	1
0	1
0	1
0	1
0	1
0	1
0	1

 Byte 1

7	0
1	0
0	0
1	0
0	0
0	0
0	0
0	0

 Byte 2

Before execution

Accumulator

1	1	1	0	0	1	0	1

7 0

Port 1

1	0	1	0	1	1	1	1

7 0

After execution

Accumulator

1	0	1	0	0	1	0	1

7 0

Port 1

1	0	1	0	1	1	1	1

7 0

## 15. ANL C,bit address (Logical AND bit to carry flag)

Instruction code : 

7	0
1	0 0 0 0 0 0 1 0

 Byte 1

Bit address : 

7	0
b <sub>7</sub>	b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>

 Byte 2

Operation : (C) ← (C) AND (bit address)

Number of bytes : 2

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

•							
---	--	--	--	--	--	--	--

Description : The logical AND between the carry flag and the specified bit address content is determined. The result is stored in the carry flag.

Example ANL C, ACC.5

Instruction code : 

7	0
1	0 0 0 0 0 0 1 0

 Byte 1

7	0
1	1 1 1 0 0 1 0 1

 Byte 2

Before execution

Carry flag

1
---

Accumulator

7	5	0
1	0 0 1 1	0 1 0

After execution

Carry flag

0
---

Accumulator

7	5	0
1	0 0 1 1	0 1 0

16. ANL C,/bit address (Logical AND complement bit to carry flag)

Instruction code : 

7	0
1	0
1	1
0	0
0	0
0	0

 Byte 1

Bit address : 

7	0
b <sub>7</sub>	b <sub>6</sub>
b <sub>5</sub>	b <sub>4</sub>
b <sub>3</sub>	b <sub>2</sub>
b <sub>1</sub>	b <sub>0</sub>

 Byte 2

Operation :  $(C) \leftarrow (C) \text{ AND } (\text{bit address})$

Number of bytes : 2

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

•							
---	--	--	--	--	--	--	--

Description : The logical AND between the carry flag and the complement of specified bit address content is determined. The result is stored in the carry flag.

Example ANL C,/P1.3

Instruction code : 

7	0
1	0
1	1
0	0
0	0
0	0

 Byte 1

7	0
1	0
0	1
0	0
1	1

 Byte 2

Before execution

After execution

Carry flag

Carry flag

1
---

0
---

Port 1

Port 1

7	3	0
0	0	0
0	0	0
1	0	1
0	1	0

7	3	0
0	0	0
0	0	0
1	0	1
0	1	0

7

17. ANL data address, #data (Logical AND immediate data to memory)

Instruction code : 

7	0
0	1
0	1
0	0
1	1

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub>
a <sub>5</sub>	a <sub>4</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

#data : 

7	0
l <sub>7</sub>	l <sub>6</sub>
l <sub>5</sub>	l <sub>4</sub>
l <sub>3</sub>	l <sub>2</sub>
l <sub>1</sub>	l <sub>0</sub>

 Byte 3

Operation : (data address) ← (data address) AND #data

Number of bytes : 3

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The logical AND between an 8-bit immediate data value and the specified data address contents is determined. The result is stored in the specified data address.

Example ANL DPH, #0AAH

Instruction code : 

7	0
0	1
0	1
0	0
1	1

 Byte 1

7	0
1	0
0	0
0	0
1	1

 Byte 2

7	0
1	0
1	0
1	0
1	0

 Byte 3

Before execution

DPH 

7	0
1	1
1	1
1	1
1	1
1	1
1	1

After execution

DPH 

7	0
1	0
1	0
1	0
1	0
1	0
1	0

18. ANL data address, A (Logical AND accumulator to memory)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0 1 0 1 0 0 1 0</div> </div>	Byte 1
Data address	:	<div> <div>7</div> <div>0</div> <div>a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub></div> </div>	Byte 2
Operation	:	(data address) ← (data address) AND (A)	
Number of bytes	:	2	
Number of cycles	:	1	
Flags	:	C AC F0 RS1 RS0 OV F1 P	
(PSW)	:	<div> <div>7</div> <div>0</div> <div></div> </div>	

Description : The logical AND between the accumulator and the specified data address contents is determined. The result is stored in the specified data address.

Example ANL TCON, A

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0 1 0 1 0 0 1 0</div> </div>	Byte 1
	:	<div> <div>7</div> <div>0</div> <div>1 0 0 0 1 0 0 0</div> </div>	Byte 2
Before execution			
Accumulator	:	<div> <div>7</div> <div>0</div> <div>0 1 0 1 1 0 1 0</div> </div>	
TCON	:	<div> <div>7</div> <div>0</div> <div>1 0 1 1 0 1 0 1</div> </div>	
After execution			
Accumulator	:	<div> <div>7</div> <div>0</div> <div>0 1 0 1 1 0 1 0</div> </div>	
TCON	:	<div> <div>7</div> <div>0</div> <div>0 0 0 1 0 0 0 0</div> </div>	

**19. CJNE @Rr, #data, code address**  
**(Compare indirect address to immediate data, jump if not equal)**

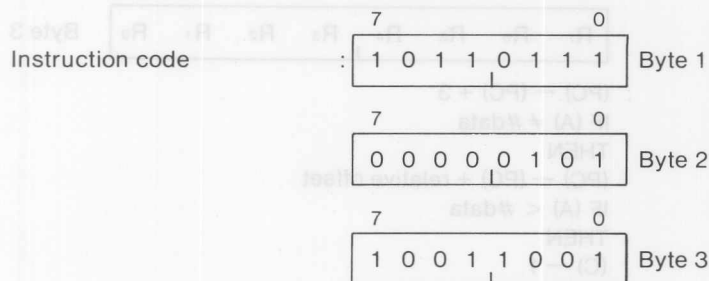
Instruction code	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>r</td></tr></table>	7	6	5	4	3	2	1	0	1	0	1	1	0	1	1	r	Byte 1					
7	6	5	4	3	2	1	0																	
1	0	1	1	0	1	1	r																	
# data	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>I<sub>7</sub></td><td>I<sub>6</sub></td><td>I<sub>5</sub></td><td>I<sub>4</sub></td><td>I<sub>3</sub></td><td>I<sub>2</sub></td><td>I<sub>1</sub></td><td>I<sub>0</sub></td></tr></table>	7	6	5	4	3	2	1	0	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>	Byte 2					
7	6	5	4	3	2	1	0																	
I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>																	
Relative offset	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>R<sub>7</sub></td><td>R<sub>6</sub></td><td>R<sub>5</sub></td><td>R<sub>4</sub></td><td>R<sub>3</sub></td><td>R<sub>2</sub></td><td>R<sub>1</sub></td><td>R<sub>0</sub></td></tr></table>	7	6	5	4	3	2	1	0	R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>	Byte 3					
7	6	5	4	3	2	1	0																	
R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>																	
Operation	:	$(PC) \leftarrow (PC) + 3$ IF $((Rr) \neq \#data)$ $r = 0$ or $1$ THEN $(PC) \leftarrow (PC) + \text{relative offset}$ IF $((Rr) < \#data)$ $r = 0$ or $1$ THEN $(C) \leftarrow 1$ ELSE $(C) \leftarrow 0$																						
Number of bytes	:	3																						
Number of cycles	:	2																						
Flags	:	<table><tr><td>C</td><td>AC</td><td>F0</td><td>RS1</td><td>RS0</td><td>OV</td><td>F1</td><td>P</td></tr><tr><td>●</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							C	AC	F0	RS1	RS0	OV	F1	P	●							
C	AC	F0	RS1	RS0	OV	F1	P																	
●																								
(PSW)																								

## Description

: The data memory location contents addressed by the register r contents are compared with an immediate data value. Control is shifted to a relative jump address if the compared data is not equal. If the compared data is equal, control is shifted to the next address following this instruction. The carry flag is set to 1 if the immediate data value is greater than the specified address contents, but is set to 0 if otherwise.

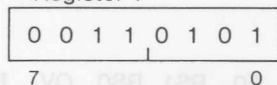
Example CJNE @R1, #05H, TEST

LOC	OBJ	SOURCE
00B4	2155	TEST : AJMP TEST1
0118	B70599	COMP : CJNE @R1, #05H, TEST
011B	020500	OUT: LJMP OUT1

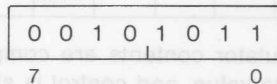


Before execution

Register 1



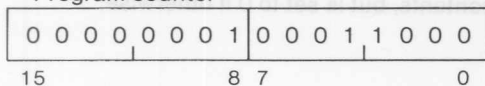
35H



Carry flag

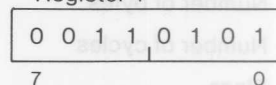


Program counter

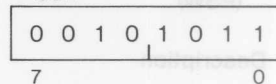


After execution

Register 1



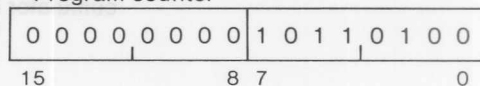
35H



Carry flag



Program counter





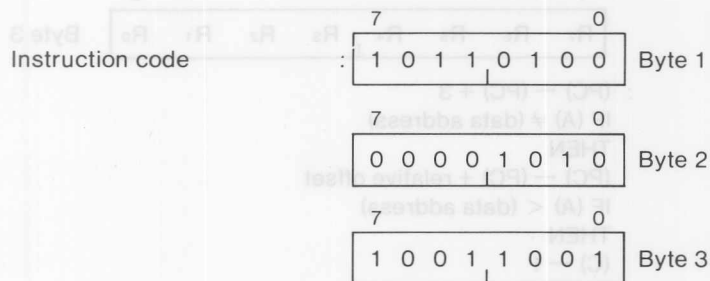
20. CJNE A, #data, code address

(Compare immediate data to accumulator, jump if not equal)

Instruction code	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	7							0	1	0	1	1	0	1	0	0	Byte 1					
7							0																	
1	0	1	1	0	1	0	0																	
# data	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>I<sub>7</sub></td><td>I<sub>6</sub></td><td>I<sub>5</sub></td><td>I<sub>4</sub></td><td>I<sub>3</sub></td><td>I<sub>2</sub></td><td>I<sub>1</sub></td><td>I<sub>0</sub></td></tr></table>	7							0	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>	Byte 2					
7							0																	
I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>																	
Relative offset	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>R<sub>7</sub></td><td>R<sub>6</sub></td><td>R<sub>5</sub></td><td>R<sub>4</sub></td><td>R<sub>3</sub></td><td>R<sub>2</sub></td><td>R<sub>1</sub></td><td>R<sub>0</sub></td></tr></table>	7							0	R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>	Byte 3					
7							0																	
R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>																	
Operation	:	$(PC) \leftarrow (PC) + 3$ IF (A) $\neq$ #data THEN $(PC) \leftarrow (PC) + \text{relative offset}$ IF (A) $<$ #data THEN (C) $\leftarrow$ 1 ELSE (C) $\leftarrow$ 0																						
Number of bytes	:	3																						
Number of cycles	:	2																						
Flags (PSW)	:	<table><tr><td>C</td><td>AC</td><td>F0</td><td>RS1</td><td>RS0</td><td>OV</td><td>F1</td><td>P</td></tr><tr><td>●</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							C	AC	F0	RS1	RS0	OV	F1	P	●							
C	AC	F0	RS1	RS0	OV	F1	P																	
●																								
Description	:	The accumulator contents are compared with an im																						

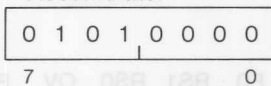
Example CJNE A, #0AH, SS1

LOC	OBJ	SOURCE
0064	FF	SS1 : MOV R7, A
00C8	B40599	COMP : CJNE A, #0AH, SS1
00CB	0D	INCR: INC R5



Before execution

Accumulator

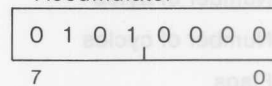


Carry flag



After execution

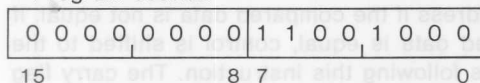
Accumulator



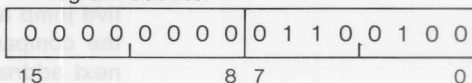
Carry flag



Program counter



Program counter



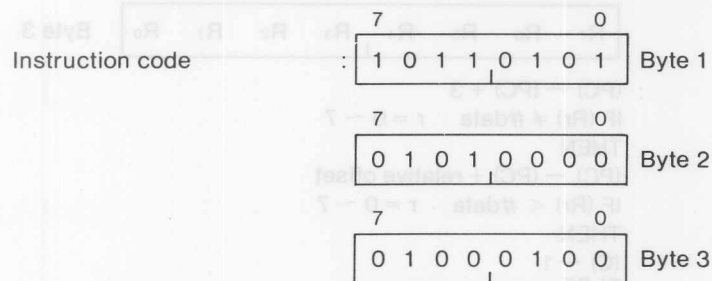
## 21. CJNE A, data address, code address

(Compare memory to accumulator, jump if not equal)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>1 0 1 1 0 1 0 1</div> </div>	Byte 1
Data address	:	<div> <div>7</div> <div>0</div> <div>a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub></div> </div>	Byte 2
Relative offset	:	<div> <div>7</div> <div>0</div> <div>R<sub>7</sub> R<sub>6</sub> R<sub>5</sub> R<sub>4</sub> R<sub>3</sub> R<sub>2</sub> R<sub>1</sub> R<sub>0</sub></div> </div>	Byte 3
Operation	:	(PC) ← (PC) + 3 IF (A) ≠ (data address) THEN (PC) ← (PC) + relative offset IF (A) < (data address) THEN (C) ← 1 ELSE (C) ← 0	
Number of bytes	:	3	
Number of cycles	:	2	
Flags	:	C   AC   F0   RS1   RS0   OV   F1   P	
(PSW)	:	<div> <div>●</div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>	
Description	:	The accumulator contents are compared with the specified data address, and control is shifted to a relative jump address if the compared data is not equal. If the compared data is equal, control is shifted to the next address following this instruction. The carry flag is set to 1 if the specified data address contents are greater than the accumulator contents, but is set to 0 if otherwise.	

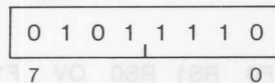
Example CJNE A, 50H, NEXT

LOC	OBJ	SOURCE
10DC	B55044	COMP : CJNE A, 50H, NEXT
10DF	120100	CAL : LCALL TEST
.....	.....	.....
1123	14	NEXT : DEC A
.....	.....	.....

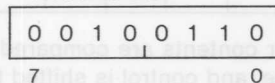


Before execution

50H



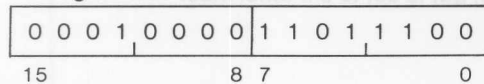
Accumulator



Carry flag

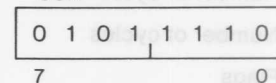


Program counter

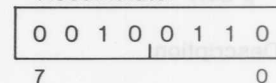


After execution

50H



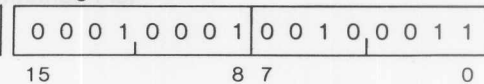
Accumulator



Carry flag



Program counter

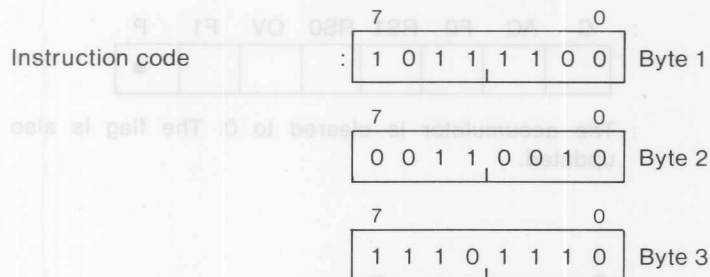


**22. CJNE Rr, #data, code address****(Compare immediate data to register, jump if not equal)**

Instruction code	:	<div> <div>7</div> <div>0</div> <div>1</div> <div>0</div> <div>1</div> <div>1</div> <div>1</div> <div><math>r_2</math></div> <div><math>r_1</math></div> <div><math>r_0</math></div> </div>	Byte 1
#data	:	<div> <div>7</div> <div>0</div> <div><math>I_7</math></div> <div><math>I_6</math></div> <div><math>I_5</math></div> <div><math>I_4</math></div> <div><math>I_3</math></div> <div><math>I_2</math></div> <div><math>I_1</math></div> <div><math>I_0</math></div> </div>	Byte 2
Relative offset	:	<div> <div>7</div> <div>0</div> <div><math>R_7</math></div> <div><math>R_6</math></div> <div><math>R_5</math></div> <div><math>R_4</math></div> <div><math>R_3</math></div> <div><math>R_2</math></div> <div><math>R_1</math></div> <div><math>R_0</math></div> </div>	Byte 3
Operation	:	$(PC) \leftarrow (PC) + 3$ IF $(Rr) \neq \#data$ $r = 0 \sim 7$ THEN $(PC) \leftarrow (PC) + \text{relative offset}$ IF $(Rr) < \#data$ $r = 0 \sim 7$ THEN $(C) \leftarrow 1$ ELSE $(C) \leftarrow 0$	
Number of bytes	:	3	
Number of cycles	:	2	
Flags	:	<div> <div>C</div> <div>AC</div> <div>F0</div> <div>RS1</div> <div>RS0</div> <div>OV</div> <div>F1</div> <div>P</div> </div>	
(PSW)	:	<div> <div>•</div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>	
Description	:	The register $r$ contents are compared with an immediate data value, and control is shifted to a relative jump address if the compared data is not equal. If the compared data is equal, control is shifted to the next address following this instruction. The carry flag is set to 1 if the immediate data value is greater than the register $r$ contents, but is set to 0 if otherwise.	

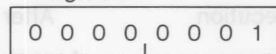
Example CJNE R4, #32H, COUNT

LOC	OBJ	SOURCE
0473	0C	COUNT: INC R4
0482	BC32EE	COMP: CJNE R4, #32H, COUNT



Before execution

Register 4

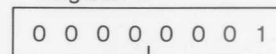


Carry flag



After execution

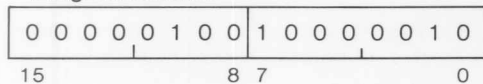
Register 4



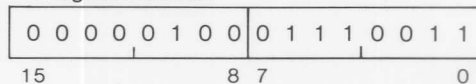
Carry flag



Program counter



Program counter



23. CLR A (Clear accumulator)

Instruction code	:	<div><div>7</div><div>1</div><div>1</div><div>1</div><div>0</div><div>0</div><div>1</div><div>0</div><div>0</div><div>0</div></div>	Byte 1
Operation	:	(A) ← 0	
Number of bytes	:	1	
Number of cycles	:	1	
Flags (PSW)	:	<div><div>C</div><div>AC</div><div>F0</div><div>RS1</div><div>RS0</div><div>OV</div><div>F1</div><div>P</div></div>	
Description	:	The accumulator is cleared to 0. The flag is also updated.	

Example CLR A

Instruction code	:	<div> <div>7</div> <div>1</div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> </div>	Byte 1
Before execution			
Accumulator			
<div> <div>7</div> <div>1</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> <div>1</div> <div>0</div> <div>1</div> <div>0</div> <div>1</div> </div>			
After execution			
Accumulator			
<div> <div>7</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> </div>			

# 24. CLR C (Clear carry flag)

Instruction code : 

7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	1

 Byte 1

Operation : (C) ← 0

Number of bytes : 1

Number of cycles : 1

Flags : 

C	AC	FO	RS1	RS0	OV	F1	P
●							

  
(PSW)

Description : The carry flag is cleared to 0.

Example CLR C

Instruction code : 

7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	1

 Byte 1

Before execution

After execution

Carry flag

Carry flag

1

0



# 25. CLR bit address (Clear bit)

Instruction code : 

7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	0

 Byte 1

Bit address : 

7	6	5	4	3	2	1	0
b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>

 Byte 2

Operation : (bit address) ← 0

Number of bytes : 2

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P

Description : The specified bit address content is cleared to 0.

Example CLR P1.5

Instruction code : 

7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	0

 Byte 1

7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	1

 Byte 2

Before execution

After execution

<p>Port 1</p> <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td></tr></table>	7	6	5	4	3	2	1	0	1	1	1	1	1	1	1	1	<p>Port 1</p> <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td></tr></table>	7	6	5	4	3	2	1	0	1	1	0	1	1	1	1	1
7	6	5	4	3	2	1	0																										
1	1	1	1	1	1	1	1																										
7	6	5	4	3	2	1	0																										
1	1	0	1	1	1	1	1																										

26. CPL A (Complement accumulator)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>1</div><div>1</div><div>1</div><div>1</div><div>0</div><div>1</div><div>0</div><div>0</div> </div>	Byte 1
Operation	:	$(A) \leftarrow \overline{(A)}$	
Number of bytes	:	1	
Number of cycles	:	1	
Flags	:	C AC FO RS1 RS0 OV F1 P	
(PSW)	:	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	

Description : The accumulator data 0 is set to 1, and 1 is set to 0.

Example CPL A

Instruction code	:	<div> <div>7</div> <div>0</div> <div>1</div><div>1</div><div>1</div><div>1</div><div>0</div><div>1</div><div>0</div><div>0</div> </div>	Byte 1
Before execution			
Accumulator			
	:	<div> <div>7</div> <div>0</div> <div>1</div><div>1</div><div>0</div><div>1</div><div>0</div><div>1</div><div>0</div><div>1</div> </div>	
After execution			
Accumulator			
	:	<div> <div>7</div> <div>0</div> <div>0</div><div>0</div><div>1</div><div>0</div><div>1</div><div>0</div><div>1</div><div>0</div> </div>	

# 27. CPL C (Complement carry flag)

Instruction code : 

7	0
1	0
1	1
0	0
1	1

 Byte 1

Operation :  $(C) \leftarrow \overline{(C)}$

Number of bytes : 1

Number of cycles : 1

Flags : 

C	AC	F0	RS1	RS0	OV	F1	P
●							

  
(PSW)

Description : The carry flag is set to 1 if 0, and set to 0 if 1.

Example CPL C

Instruction code : 

7	0
1	0
1	1
0	0
1	1

 Byte 1

	Before execution	After execution
Carry flag	<div>1</div>	<div>0</div>
Carry flag	<div>0</div>	<div>1</div>

28. CPL bit address (Complement bit)

Instruction code : 

7	0
1	0
1	1
0	0
1	0

 Byte 1

Bit address : 

7	0
b <sub>7</sub>	b <sub>6</sub>
b <sub>5</sub>	b <sub>4</sub>
b <sub>3</sub>	b <sub>2</sub>
b <sub>1</sub>	b <sub>0</sub>

 Byte 2

Operation : (bit address)  $\leftarrow$   $\overline{\text{(bit address)}}$

Number of bytes : 2

Number of cycles : 1

Flags : 

C	AC	F0	RS1	RS0	OV	F1	P

  
(PSW)

Description : The specified bit address content is set to 1 if 0, and set to 0 if 1.

Example CPL B.7

Instruction code : 

7	0
1	0
1	1
0	0
1	0

 Byte 1

7	0
1	1
1	1
0	1
1	1

 Byte 2

Before execution

B register  

0	1	0	1	0	1	1	1
7							0

After execution

B register  

1	1	0	1	0	1	1	1
7							0

29. DA A (Decimal adjust accumulator)

Instruction code : 

7	6	5	4	3	2	1	0
1	1	0	1	0	1	0	0

 Byte 1

Operation :  $10^0 + 6 \leftarrow (AC) = 1$  or  $10_0 > 10$   
 $10^1 + 6 \leftarrow (C) = 1$  or  $10^1 > 10$   
 $(C) \leftarrow 1$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

●	●	●	●	●	●	●	●
---	---	---	---	---	---	---	---

Description : The arithmetic operation result located in the accumulator following an addition between two 2-digit decimal numbers is converted to a normal decimal number. When the contents of accumulator bits 0 thru 3 ( $10^0$  digit) are greater than 9, or when the auxiliary carry (AC) is 1, 6 is added to accumulator bit 0 thru 3. And if the contents of accumulator bits 4 thru 7 ( $10^1$  digit) exceed 9, or if the result obtained by adding a carry from the lower order digits after compensation is greater than 9, or if the carry flag is 1, 6 is added to the data in accumulator bits 4 thru 7. The flag is also updated.

Example DA A

Instruction code : 1 1 0 1 0 1 0 0 0 Byte 1

Before execution

After execution

Accumulator

Accumulator

1 0 1 1 0 1 0 1

0 0 0 1 0 1 0 1

C

AC

C

AC

0

0

1

0

Before execution

After execution

Accumulator

Accumulator

0 0 1 1 0 0 0 1

1 0 0 1 0 1 1 1

C

AC

C

AC

1

1

1

1

Before execution

After execution

Accumulator

Accumulator

1 0 0 1 1 1 0 0

0 0 0 0 0 0 1 0

C

AC

C

AC

0

0

1

0

7

### 30. DEC @Rr (Decrement indirect address)

Instruction code : 

7	0
0	0
0	0
1	0
1	1
r	r

 Byte 1

Operation :  $((Rr)) \leftarrow ((Rr)) - 1$   $r = 0$  or  $1$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P

--	--	--	--	--	--	--	--

Description : The contents of the data memory location addressed by the register r contents are decremented by 1.

Example DEC @R0

Instruction code : 

7	0
0	0
0	1
0	1
1	1
0	0

 Byte 1

Before execution

Register 0

0	1	1	0	1	0	1	0
7							0

6AH

1	0	0	1	0	0	0	0
7							0

After execution

Register 0

0	1	1	0	1	0	1	0
7							0

6AH

1	0	0	0	1	1	1	1
7							0

31. DEC A (Decrement accumulator)

Instruction code : 

7	0
0	0
0	1
0	0

 Byte 1

Operation :  $(A) \leftarrow (A) - 1$

Number of bytes : 1

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
							•

Description : The accumulator contents are decremented by 1, and the flag is undated.

Example DEC A

Instruction code : 

7	0
0	0
0	1
0	0

Before execution

Accumulator

7	0
1	0
1	0
0	1
0	0
0	0

After execution

Accumulator

7	0
1	0
1	0
0	0
1	1
1	1



### 32. DEC Rr (Decrement register)

Instruction code : 

7	0
0	0
0	1
1	1
$r_2$	$r_1$
$r_0$	$r_0$

 Byte 1

Operation :  $(Rr) \leftarrow (Rr) - 1 \quad r = 0 \sim 7$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--	--

Description : The register r contents are decremented by 1.

Example DEC R7

Instruction code : 

7	0
0	0
0	1
1	1
1	1
1	1

 Byte 1

Before execution

Register 7

1	0	0	0	0	0	0	0
7							0

After execution

Register 7

0	1	1	1	1	1	1	1
7							0

### 33. DEC data address (Decrement memory)

Instruction code : 

7	0
0	0
0	1
0	0
1	0
1	1
0	0
1	1

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub>
a <sub>5</sub>	a <sub>4</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (data address) ← (data address) - 1

Number of bytes : 2

Number of cycles : 1

Flags : 

C	AC	F0	RS1	RS0	OV	F1	P

  
(PSW)

Description : The specified data address contents are decremented by 1.

Example DEC 5AH

Instruction code : 

7	0
0	0
0	1
0	1
0	1
0	1
0	1
0	1

 Byte 1

7	0
0	1
0	1
1	1
0	1
0	1
0	1
0	0

 Byte 2

Before execution

After execution

5AH

5AH

7	0
1	1
1	1
1	1
1	1
1	1
1	1
1	1

7	0
1	1
1	1
1	1
1	1
1	1
1	1
1	0

34. DIV AB (Divide accumulator by B)

Instruction code	:	<div><div>7</div><div>0</div><div>1 0 0 0 0 1 0 0</div></div>	Byte 1	
Operation	:	(A) quotient $\leftarrow$ (A)/(B) (B) remainder		
Number of bytes	:	1		
Number of cycles	:	4		
Flags	:	C AC F0 RS1 RS0 OV F1 P		
(PSW)		<div><div>•</div><div></div><div></div><div></div><div></div><div>•</div><div></div><div>•</div></div>		
Description	:	The accumulator contents are divided by the contents of arithmetic operation register (B). The two data values are handled as integers without sign. The quotient is placed in the accumulator, and the remainder in the arithmetic operation register (B). The carry flag is always cleared, and the overflow flag (OV) is set to 1 if a division by 0 is executed. This flag is cleared in all other cases. In case of division by 0, the contents of the accumulator and the arithmetic operation register (B) remain unchanged.		

Example    DIV AB (0AEH ÷ 7H = 18H ..... remainder 6H)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>1 0 0 0 0 1 0 0</div> </div>	Byte 1
		Before execution	After execution
		Accumulator	Accumulator
		<div> <div>7</div> <div>0</div> <div>1 0 1 0 1 1 1 0</div> </div>	<div> <div>7</div> <div>0</div> <div>0 0 0 1 1 0 0 0</div> </div>
		B register	B register
		<div> <div>7</div> <div>0</div> <div>0 0 0 0 0 1 1 1</div> </div>	<div> <div>7</div> <div>0</div> <div>0 0 0 0 0 1 1 0</div> </div>

35. DJNZ Rr, Code address (Decrement register, and jump if not zero)

Instruction code	:	<div><div>7</div><div>0</div><div>1</div><div>1</div><div>0</div><div>1</div><div>1</div><div>r<sub>2</sub></div><div>r<sub>1</sub></div><div>r<sub>0</sub></div></div>	Byte 1						
Relative offset	:	<div><div>7</div><div>0</div><div>R<sub>7</sub></div><div>R<sub>6</sub></div><div>R<sub>5</sub></div><div>R<sub>4</sub></div><div>R<sub>3</sub></div><div>R<sub>2</sub></div><div>R<sub>1</sub></div><div>R<sub>0</sub></div></div>	Byte 2						
Operation	:	$(PC) \leftarrow (PC) + 2$ $(Rr) \leftarrow (Rr) - 1 \quad r = 0 \sim 7$ IF $(Rr) \neq 0$ THEN $(PC) \leftarrow (PC) + \text{relative offset}$							
Number of bytes	:	2							
Number of cycles	:	2							
Flags	:	C	AC	F0	RS1	RS0	OV	F1	P
(PSW)	:								

Description : The register r contents are decremented by 1. Control is shifted to a relative jump address if the register r contents are not 0 as a result of the decrement. Control is shifted to the next address following this instruction if the result is 0.

Example DJNZ R1, LOOP

LOC	OBJ	SOURCE
00FE	2F	LOOP : ADD A, R7
010B	D9F1	COUNT : DJNZ R1, LOOP

Instruction code

7	0
: 1 1 0 1 1 0 0 1	
Byte 1	
7	0
1 1 1 1 0 0 0 1	
Byte 2	

Before execution

Register 1

0	0	0	0	1	0	0	0
7							0

After execution

Register 1

0	0	0	0	0	1	1	1
7							0

Program counter

0	0	0	0	0	0	0	1
15							8
							7
							0

Program counter

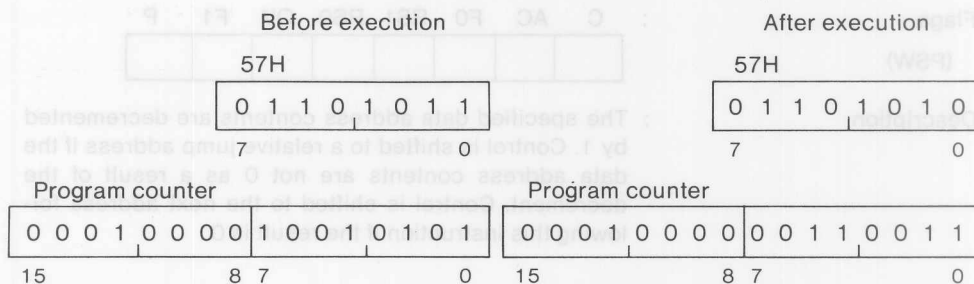
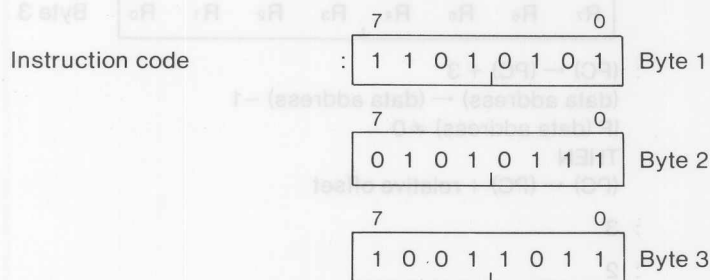
0	0	0	0	0	0	0	0
15							8
							7
							0

36. DJNZ data address, Code address (Decrement memory, and jump if not zero)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> <div>1</div> <div>0</div> <div>1</div> </div>	Byte 1
Data address	:	<div> <div>7</div> <div>0</div> <div>a<sub>7</sub></div> <div>a<sub>6</sub></div> <div>a<sub>5</sub></div> <div>a<sub>4</sub></div> <div>a<sub>3</sub></div> <div>a<sub>2</sub></div> <div>a<sub>1</sub></div> <div>a<sub>0</sub></div> </div>	Byte 2
Relative offset	:	<div> <div>7</div> <div>0</div> <div>R<sub>7</sub></div> <div>R<sub>6</sub></div> <div>R<sub>5</sub></div> <div>R<sub>4</sub></div> <div>R<sub>3</sub></div> <div>R<sub>2</sub></div> <div>R<sub>1</sub></div> <div>R<sub>0</sub></div> </div>	Byte 3
Operation	:	$(PC) \leftarrow (PC) + 3$ $(data\ address) \leftarrow (data\ address) - 1$ IF (data address) $\neq 0$ THEN $(PC) \leftarrow (PC) + relative\ offset$	
Number of bytes	:	3	
Number of cycles	:	2	
Flags (PSW)	:	<div> <div>C</div> <div>AC</div> <div>F0</div> <div>RS1</div> <div>RS0</div> <div>OV</div> <div>F1</div> <div>P</div> </div> <div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>	
Description	:	The specified data address contents are decremented by 1. Control is shifted to a relative jump address if the data address contents are not 0 as a result of the decrement. Control is shifted to the next address following this instruction if the result is 0.	

Example DJNZ 57H, LOOP 1

LOC	OBJ	SOURCE
1033	A957	LOOP 1 : MOV R1, 57H
1095	D5579B	COUNT : DJNZ 57H, LOOP 1



37. INC @Rr (Increment indirect address)

Instruction code : 

7	0
0	0
0	0
0	0
0	1
1	1
r	r

 Byte 1

Operation :  $((Rr)) \leftarrow ((Rr)) + 1$  r = 0 or 1

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--	--	--

Description : The contents of the data memory location addressed by the register r contents are incremented by 1.

Example INC @R1

Instruction code : 

7	0
0	0
0	0
0	0
0	1
1	1
1	1

 Byte 1

	Before execution	After execution																																
Register 1	<table><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>7</td><td colspan="6"></td><td>0</td></tr></table>	0	1	1	0	0	1	0	1	7							0	<table><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>7</td><td colspan="6"></td><td>0</td></tr></table>	0	1	1	0	0	1	0	1	7							0
0	1	1	0	0	1	0	1																											
7							0																											
0	1	1	0	0	1	0	1																											
7							0																											
65H	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>7</td><td colspan="6"></td><td>0</td></tr></table>	0	0	0	0	1	1	1	1	7							0	<table><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>7</td><td colspan="6"></td><td>0</td></tr></table>	0	0	0	1	0	0	0	0	7							0
0	0	0	0	1	1	1	1																											
7							0																											
0	0	0	1	0	0	0	0																											
7							0																											



### 38. INC A (Increment accumulator)

Instruction code : 

7	0
0	0
0	0
0	0
0	1
0	0
0	0

 Byte 1

Operation :  $(A) \leftarrow (A) + 1$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

								●
--	--	--	--	--	--	--	--	---

Description : The accumulator contents are incremented by 1, and the flag is updated.

Example INC A

Instruction code : 

7	0
0	0
0	0
0	0
0	1
0	0
0	0

 Byte 1

Before execution		After execution
Accumulator		Accumulator
<div>10110111</div> <div>70</div>		<div>10111000</div> <div>70</div>

39. INC DPTR (Increment data pointer)

Instruction code : 

7	0
1	0
1	0
0	0
0	0
1	1

 Byte 1

Operation :  $(DPTR) \leftarrow (DPTR) + 1$

Number of bytes : 1

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : 16-bit contents of data pointer (DPH·DPL) are incremented by 1.

Example INC DPTR

Instruction code : 

7	0
1	0
1	0
0	0
0	0
1	1

 Byte 1

Before execution

DPH								DPL							
0	1	1	0	1	0	0	0	1	1	1	1	1	1	1	1
15								0							

After execution

DPH								DPL							
0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0
15								0							

#### 40. INC Rr (Increment register)

Instruction code : 

7	0
0	0
0	0
0	0
1	r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>

 Byte 1

Operation :  $(Rr) \leftarrow (Rr) + 1 \quad r = 0 \sim 7$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--	--	--

Description : The register r contents are incremented by 1.

Example INC R5

Instruction code : 

7	0
0	0
0	0
0	0
1	1 0 1

 Byte 1

Before execution

Register 5

7	0
1	0
1	1
1	1
1	1
1	1
1	1
1	1

After execution

Register 5

7	0
1	1
0	0
0	0
0	0
0	0
0	0
0	0

41. INC data address (Increment memory)

Instruction code : 

7	0
0	0
0	0
0	0
0	1
0	1

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub>
a <sub>5</sub>	a <sub>4</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (data address)  $\leftarrow$  (data address) + 1

Number of bytes : 2

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
---	----	----	-----	-----	----	----	---

Description : The specified data address contents are incremented by 1.

Example INC P1

Instruction code : 

7	0
0	0
0	0
0	0
0	1
0	1

 Byte 1

Data address : 

7	0
1	0
0	1
0	0
0	0
0	0

 Byte 2

Before execution

Port 1

7	0
0	0
0	0
0	1
1	1
1	1

After execution

Port 1

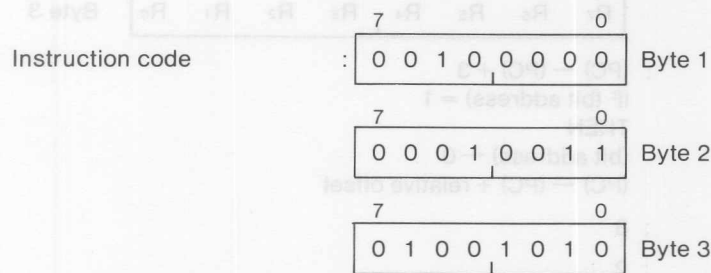
7	0
0	0
0	0
1	0
0	0
0	0

42. JB bit address, Code address (Jump if bit is set)

Instruction code	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	0	0	1	0	0	0	0	0	Byte 1
7	6	5	4	3	2	1	0												
0	0	1	0	0	0	0	0												
Bit address	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>b<sub>7</sub></td><td>b<sub>6</sub></td><td>b<sub>5</sub></td><td>b<sub>4</sub></td><td>b<sub>3</sub></td><td>b<sub>2</sub></td><td>b<sub>1</sub></td><td>b<sub>0</sub></td></tr></table>	7	6	5	4	3	2	1	0	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2
7	6	5	4	3	2	1	0												
b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>												
Relative offset	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>R<sub>7</sub></td><td>R<sub>6</sub></td><td>R<sub>5</sub></td><td>R<sub>4</sub></td><td>R<sub>3</sub></td><td>R<sub>2</sub></td><td>R<sub>1</sub></td><td>R<sub>0</sub></td></tr></table>	7	6	5	4	3	2	1	0	R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>	Byte 3
7	6	5	4	3	2	1	0												
R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>												
Operation	:	(PC) ← (PC) + 3 IF (bit address) = 1 THEN (PC) ← (PC) + relative offset																	
Number of bytes	:	3																	
Number of cycles	:	2																	
Flags	:	C	AC	F0	RS1	RS0	OV	F1	P										
(PSW)	:	<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>																	
Description	:	Control is shifted to a relative jump address if the specified bit address content is 1. Control is shifted to the next address following this instruction if the content is 0.																	

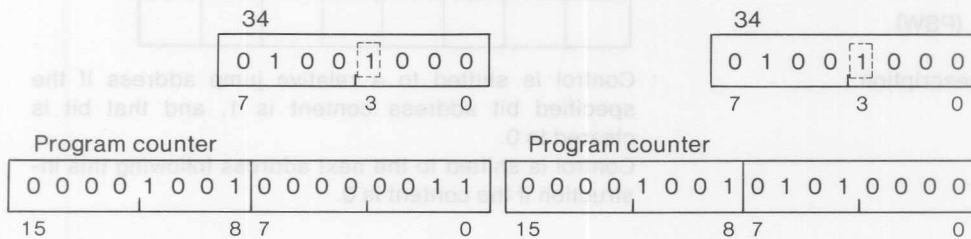
Example JB 34.3, ENTER

LOC	OBJ	SOURCE
0903	20134A	BITTS : JB 34.3, ENTER
0950	ACAO	ENTER : MOV R4, 0A0H



Before execution

After execution

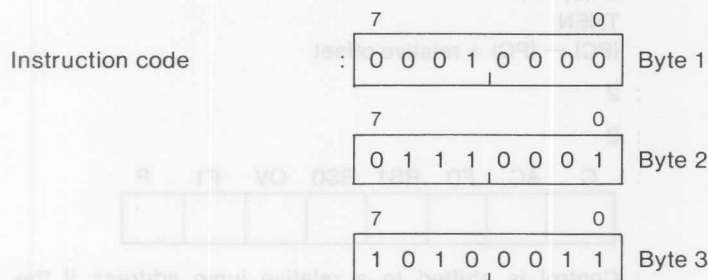


## 43. JBC bit address, Code address (Jump and clear if bit is set)

Instruction code	:	<table><tr><td>7</td><td>0</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table> Byte 1	7	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
7	0																		
0	0																		
0	1																		
0	0																		
0	0																		
0	0																		
0	0																		
0	0																		
Bit address	:	<table><tr><td>7</td><td>0</td></tr><tr><td>b<sub>7</sub></td><td>b<sub>6</sub></td></tr><tr><td>b<sub>5</sub></td><td>b<sub>4</sub></td></tr><tr><td>b<sub>3</sub></td><td>b<sub>2</sub></td></tr><tr><td>b<sub>1</sub></td><td>b<sub>0</sub></td></tr></table> Byte 2	7	0	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>							
7	0																		
b <sub>7</sub>	b <sub>6</sub>																		
b <sub>5</sub>	b <sub>4</sub>																		
b <sub>3</sub>	b <sub>2</sub>																		
b <sub>1</sub>	b <sub>0</sub>																		
Relative offset	:	<table><tr><td>7</td><td>0</td></tr><tr><td>R<sub>7</sub></td><td>R<sub>6</sub></td></tr><tr><td>R<sub>5</sub></td><td>R<sub>4</sub></td></tr><tr><td>R<sub>3</sub></td><td>R<sub>2</sub></td></tr><tr><td>R<sub>1</sub></td><td>R<sub>0</sub></td></tr></table> Byte 3	7	0	R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>							
7	0																		
R <sub>7</sub>	R <sub>6</sub>																		
R <sub>5</sub>	R <sub>4</sub>																		
R <sub>3</sub>	R <sub>2</sub>																		
R <sub>1</sub>	R <sub>0</sub>																		
Operation	:	$(PC) \leftarrow (PC) + 3$ IF (bit address) = 1 THEN (bit address) $\leftarrow$ 0 $(PC) \leftarrow (PC) + \text{relative offset}$																	
Number of bytes	:	3																	
Number of cycles	:	2																	
Flags	:	<table><tr><td>C</td><td>AC</td><td>F0</td><td>RS1</td><td>RS0</td><td>OV</td><td>F1</td><td>P</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		C	AC	F0	RS1	RS0	OV	F1	P								
C	AC	F0	RS1	RS0	OV	F1	P												
(PSW)	:																		
Description	:	Control is shifted to a relative jump address if the specified bit address content is 1, and that bit is cleared to 0. Control is shifted to the next address following this instruction if the content is 0.																	

Example JBC 46.1, COUNT 4

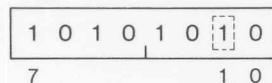
LOC	OBJ	SOURCE
00DC	C281	COUNT 4 : CLR 128.1
0136	1071A3	BTEST : JBC 46.1, COUNT 4



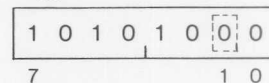
Before execution

After execution

46

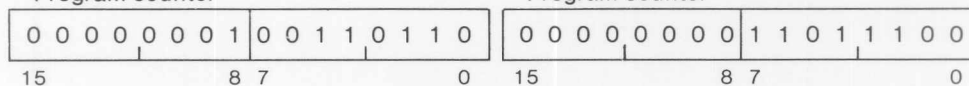


46



Program counter

Program counter



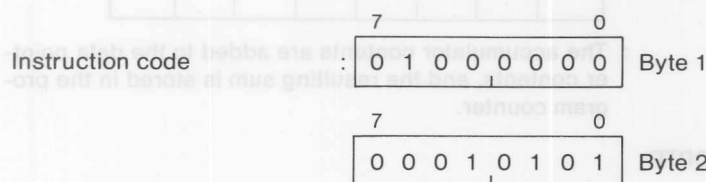


44. JC Code address (Jump if carry is set)

Instruction code	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	7							0	0	1	0	0	0	0	0	0	Byte 1
7							0												
0	1	0	0	0	0	0	0												
Relative offset	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>R<sub>7</sub></td><td>R<sub>6</sub></td><td>R<sub>5</sub></td><td>R<sub>4</sub></td><td>R<sub>3</sub></td><td>R<sub>2</sub></td><td>R<sub>1</sub></td><td>R<sub>0</sub></td></tr></table>	7							0	R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>	Byte 2
7							0												
R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>												
Operation	:	(PC) ← (PC) + 2 IF (C) = 1 THEN (PC) ← (PC) + relative offset																	
Number of bytes	:	2																	
Number of cycles	:	2																	
Flags (PSW)	:	C	AC	F0	RS1	RS0	OV	F1	P										
Description	:	Control is shifted to a relative jump address if the carry flag is 1. Control is shifted to the next address following this instruction if the content is 0.																	

Example JC CARRY

LOC	OBJ	SOURCE
16DC	7110	CHECK : ACALL ADDR
16DE	4015	JMPC : JC CARRY
16F5	07	CARRY : INC @R1

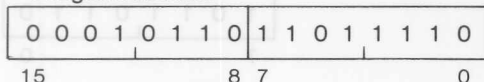


Before execution

Carry flag

1

Program counter

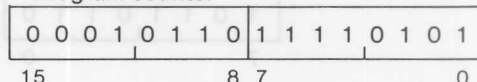


After execution

Carry flag

1

Program counter



45. **JMP @A + DPTR** (Jump to sum of accumulator and data pointer)

Instruction code : 

7	0
0	1
1	1
1	0
0	1
1	1

 Byte 1

Operation :  $(PC) \leftarrow (A) + (DPTR)$

Number of bytes : 1

Number of cycles : 2

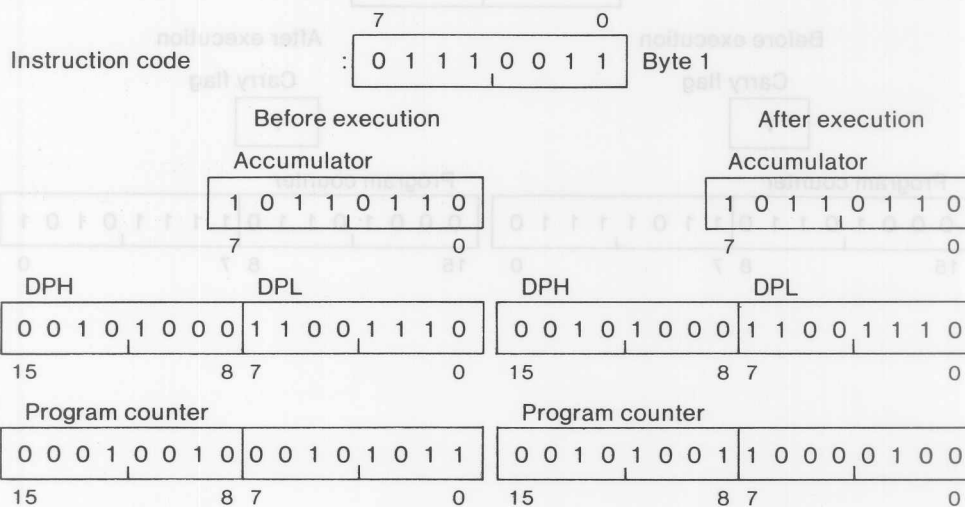
Flags : C AC F0 RS1 RS0 OV F1 P

(PSW) 

--	--	--	--	--	--	--	--

Description : The accumulator contents are added to the data pointer contents, and the resulting sum is stored in the program counter.

Example **JMP @A + DPTR**



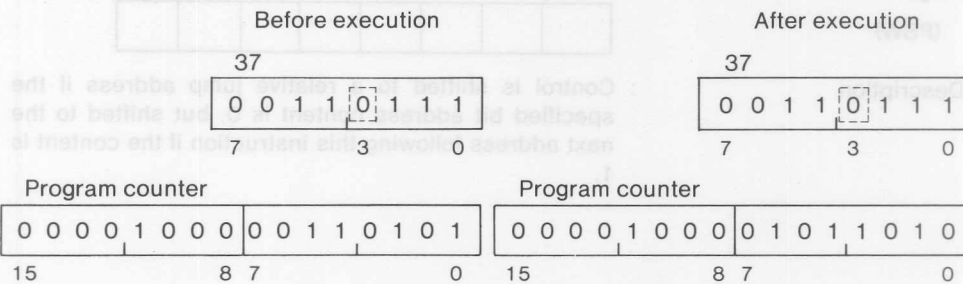
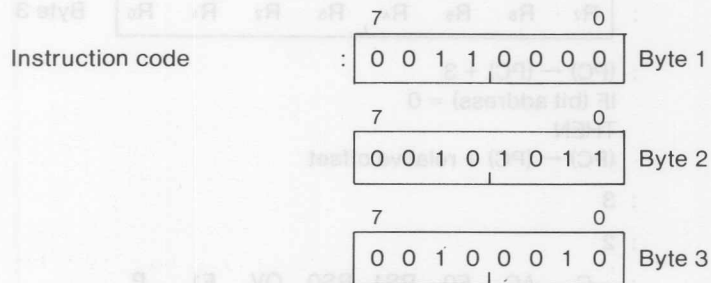
## DESCRIPTION OF INSTRUCTIONS

### 46. JNB bit address, Code address (Jump if bit is not set)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> </div>	Byte 1
Bit address	:	<div> <div>7</div> <div>0</div> <div>b<sub>7</sub></div> <div>b<sub>6</sub></div> <div>b<sub>5</sub></div> <div>b<sub>4</sub></div> <div>b<sub>3</sub></div> <div>b<sub>2</sub></div> <div>b<sub>1</sub></div> <div>b<sub>0</sub></div> </div>	Byte 2
Relative offset	:	<div> <div>7</div> <div>0</div> <div>R<sub>7</sub></div> <div>R<sub>6</sub></div> <div>R<sub>5</sub></div> <div>R<sub>4</sub></div> <div>R<sub>3</sub></div> <div>R<sub>2</sub></div> <div>R<sub>1</sub></div> <div>R<sub>0</sub></div> </div>	Byte 3
Operation	:	(PC) ← (PC) + 3 IF (bit address) = 0 THEN (PC) ← (PC) + relative offset	
Number of bytes	:	3	
Number of cycles	:	2	
Flags (PSW)	:	<div> <div>C</div> <div>AC</div> <div>F0</div> <div>RS1</div> <div>RS0</div> <div>OV</div> <div>F1</div> <div>P</div> </div>	
Description	:	Control is shifted to a relative jump address if the specified bit address content is 0, but shifted to the next address following this instruction if the content is 1.	

Example JNB 37.3, EXIT

LOC	OBJ	SOURCE
0835	302B22	TEST : JNB 37.3, EXIT
085A	E6	EXIT : MOV A, @R0

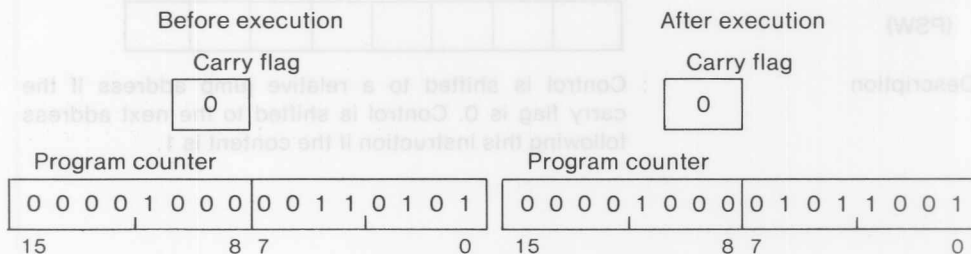
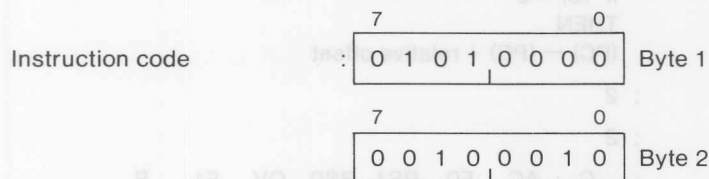


47. JNC Code address (Jump if carry is not set)

Instruction code	:	<table><tr><td>7</td><td colspan="7"></td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	7								0	0	1	0	1	0	0	0	0	Byte 1				
7								0																
0	1	0	1	0	0	0	0																	
Relative offset	:	<table><tr><td>7</td><td colspan="7"></td><td>0</td></tr><tr><td>R<sub>7</sub></td><td>R<sub>6</sub></td><td>R<sub>5</sub></td><td>R<sub>4</sub></td><td>R<sub>3</sub></td><td>R<sub>2</sub></td><td>R<sub>1</sub></td><td>R<sub>0</sub></td></tr></table>	7								0	R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>	Byte 2				
7								0																
R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>																	
Operation	:	$(PC) \leftarrow (PC) + 2$ IF (C) = 0 THEN $(PC) \leftarrow (PC) + \text{relative offset}$																						
Number of bytes	:	2																						
Number of cycles	:	2																						
Flags	:	<table><tr><td>C</td><td>AC</td><td>F0</td><td>RS1</td><td>RS0</td><td>OV</td><td>F1</td><td>P</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							C	AC	F0	RS1	RS0	OV	F1	P								
C	AC	F0	RS1	RS0	OV	F1	P																	
(PSW)																								
Description	:	Control is shifted to a relative jump address if the carry flag is 0. Control is shifted to the next address following this instruction if the content is 1.																						

Example JNC EXIT

LOC	OBJ	SOURCE
0835	5022	TEST : JNC EXIT
0859	85E0F0	EXIT : MOV B, ACC



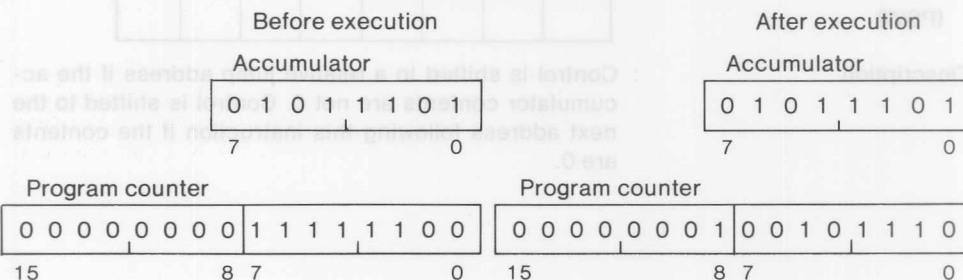
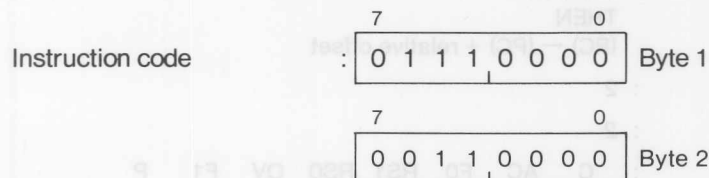
48. JNZ Code address (Jump if accumulator is not 0)

Instruction code	:	<table><tr><td>7</td><td colspan="7"></td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	7								0	0	1	1	1	0	0	0	0	Byte 1
7								0												
0	1	1	1	0	0	0	0													
Relative offset	:	<table><tr><td>7</td><td colspan="7"></td><td>0</td></tr><tr><td>R<sub>7</sub></td><td>R<sub>6</sub></td><td>R<sub>5</sub></td><td>R<sub>4</sub></td><td>R<sub>3</sub></td><td>R<sub>2</sub></td><td>R<sub>1</sub></td><td>R<sub>0</sub></td></tr></table>	7								0	R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>	Byte 2
7								0												
R <sub>7</sub>	R <sub>6</sub>	R <sub>5</sub>	R <sub>4</sub>	R <sub>3</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>0</sub>													
Operation	:	$(PC) \leftarrow (PC) + 2$ IF (A) $\neq 0$ THEN $(PC) \leftarrow (PC) + \text{relative offset}$																		
Number of bytes	:	2																		
Number of cycles	:	2																		
Flags	:	C	AC	F0	RS1	RS0	OV	F1	P											
(PSW)	:	<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>																		
Description	:	Control is shifted to a relative jump address if the accumulator contents are not 0. Control is shifted to the next address following this instruction if the contents are 0.																		



Example JNZ TEST

LOC	OBJ	SOURCE
00FC	7030	CHECK: JNZ TEST
012E	FB	TEST: MOV R3, A



49. JZ Code address (Jump if accumulator is 0)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0</div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> <div>0</div> </div>	Byte 1
Relative offset	:	<div> <div>7</div> <div>0</div> <div>R<sub>7</sub></div> <div>R<sub>6</sub></div> <div>R<sub>5</sub></div> <div>R<sub>4</sub></div> <div>R<sub>3</sub></div> <div>R<sub>2</sub></div> <div>R<sub>1</sub></div> <div>R<sub>0</sub></div> </div>	Byte 2
Operation	:	(PC) ← (PC) + 2 IF (A) = 0 THEN (PC) ← (PC) + relative offset	
Number of bytes	:	2	
Number of cycles	:	2	
Flags	:	C   AC   F0   RS1   RS0   OV   F1   P	
(PSW)	:	<div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>	
Description	:	Control is shifted to a relative jump address if the accumulator contents are 0. Control is shifted to the next address following this instruction if the contents are not 0.	

Example JZ EMPTY

LOC	OBJ	SOURCE
0099	04	EMPTY : INC A
00CA	60CD	CHECK : JZ EMPTY

Instruction code

7	0
0 1 1 0 0 0 0 0	Byte 1
7	0
1 1 0 0 1 1 0 1	Byte 2

Before execution		After execution	
Accumulator		Accumulator	
0 0 0 0 0 0 0 0		0 0 0 0 0 0 0 0	
7	0	7	0
Program counter		Program counter	
0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0		0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1	
15	8 7 0	15	8 7 0

50. LCALL Code address (Long call)

Instruction code : 

7	0
0	0
0	1
0	0
1	1
0	0

 Byte 1

Call address : 

7	0
A <sub>15</sub>	A <sub>14</sub>
A <sub>13</sub>	A <sub>12</sub>
A <sub>11</sub>	A <sub>10</sub>
A <sub>9</sub>	A <sub>8</sub>

 Byte 2

Call address : 

7	0
A <sub>7</sub>	A <sub>6</sub>
A <sub>5</sub>	A <sub>4</sub>
A <sub>3</sub>	A <sub>2</sub>
A <sub>1</sub>	A <sub>0</sub>

 Byte 3

Operation : (PC) ← (PC) + 3  
(SP) ← (SP) + 1  
((SP)) ← (PC<sub>0-7</sub>)  
(SP) ← (SP) + 1  
((SP)) ← (PC<sub>8-15</sub>)  
(PC<sub>0-15</sub>) ← A<sub>0-15</sub>

Number of bytes : 3

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The contents of the program counter (return address) are stored in the stack following an increment.  
Call address A<sub>0-15</sub> specified by operand are placed in the program counter PC<sub>0-15</sub>.  
This instruction is capable of call jump to anywhere within the entire range of 64K words.

51. LJMP Code address (Long jump)

Instruction code	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	0	0	0	0	0	0	1	0	Byte 1
7	6	5	4	3	2	1	0												
0	0	0	0	0	0	1	0												
Jump address	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>A<sub>15</sub></td><td>A<sub>14</sub></td><td>A<sub>13</sub></td><td>A<sub>12</sub></td><td>A<sub>11</sub></td><td>A<sub>10</sub></td><td>A<sub>9</sub></td><td>A<sub>8</sub></td></tr></table>	7	6	5	4	3	2	1	0	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	Byte 2
7	6	5	4	3	2	1	0												
A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>												
Jump address	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>A<sub>7</sub></td><td>A<sub>6</sub></td><td>A<sub>5</sub></td><td>A<sub>4</sub></td><td>A<sub>3</sub></td><td>A<sub>2</sub></td><td>A<sub>1</sub></td><td>A<sub>0</sub></td></tr></table>	7	6	5	4	3	2	1	0	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	Byte 3
7	6	5	4	3	2	1	0												
A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>												
Operation	:	(PC <sub>0-15</sub> ) ← A <sub>0-15</sub>																	
Number of bytes	:	3																	
Number of cycles	:	2																	
Flags	:	C	AC	F0	RS1	RS0	OV	F1	P										
(PSW)	:																		
Description	:	Jump addresses A <sub>0-15</sub> specified by operand are placed in the program counter PC <sub>0-15</sub> . This instruction is capable of jump to anywhere within the entire range of 64K words.																	

52. MOV @Rr, #data (Move immediate data to indirect address)

Instruction code : 

7	0
0	1
1	1
1	0
1	1
1	r

 Byte 1

#Data : 

7	0
I <sub>7</sub>	I <sub>6</sub>
I <sub>5</sub>	I <sub>4</sub>
I <sub>3</sub>	I <sub>2</sub>
I <sub>1</sub>	I <sub>0</sub>

 Byte 2

Operation : ((Rr)) ← #data r = 0 or 1

Number of bytes : 2

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P

Description : An 8-bit immediate data value is copied to the data memory location addressed by the register r contents.

Example MOV @R1, #0AAH

Instruction code : 

7	0
0	1
1	1
0	1
1	1
1	1

 Byte 1

7	0
1	0
1	0
1	0
1	0
1	0

 Byte 2

Before execution

Register 1

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

7

6AH

0	1	1	1	0	1	1	1
---	---	---	---	---	---	---	---

7

0

After execution

Register 1

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

7

6AH

1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---

7

0

53. MOV @Rr, A (Move accumulator to indirect address)

Instruction code : 

7	0
1	1
1	1
1	1
0	r

 Byte 1

Operation :  $((Rr)) \leftarrow (A)$   $r = 0 \text{ or } 1$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The accumulator contents are copied to the data memory location addressed by the register r contents.

Example MOV @R0, A

Instruction code : 

7	0
1	1
1	1
1	1
0	1
1	1
1	0

 Byte 1

Before execution

Register 0

0	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---

7 0

6CH

1	0	1	1	1	0	1	1
---	---	---	---	---	---	---	---

7 0

Accumulator

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

7 0

After execution

Register 0

0	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---

7 0

6CH

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

7 0

Accumulator

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

7 0

54. MOV @Rr, data address (Move memory to indirect address)

Instruction code : 

7	0
1 0 1 0	0 1 1 r

 Byte 1

Data address : 

7	0
a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub>	a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>

 Byte 2

Operation : ((Rr)) ← (data address) r = 0 or 1

Number of bytes : 2

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The specified data address contents are copied to the data memory location addressed by the register r contents.

Example MOV @R0, 0E0H

Instruction code : 

7	0
1 0 1 0 0 1 1 0	

 Byte 1  

7	0
1 1 1 0 0 0 0 0	

 Byte 2

Before execution

Accumulator

0	1	0	1	0	1	1	1
7							0

Register 0

0	1	1	1	0	0	1	0
7							0

72H

0	0	1	1	1	1	0	0
7							0

After execution

Accumulator

0	1	0	1	0	1	1	1
7							0

Register 0

0	1	1	1	0	0	1	0
7							0

72H

0	1	0	1	0	1	1	1
7							0



55. MOV A, #data (Move immediate data to accumulator)

Instruction code : 

7	0
0 1 1 1	0 1 0 0

 Byte 1

#data : 

7	0
I <sub>7</sub> I <sub>6</sub> I <sub>5</sub> I <sub>4</sub>	I <sub>3</sub> I <sub>2</sub> I <sub>1</sub> I <sub>0</sub>

 Byte 2

Operation : (A) ← #data

Number of bytes : 2

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

							●
--	--	--	--	--	--	--	---

Description : An 8-bit immediate data value is copied to the accumulator, and the flag is updated.

Example MOV A, #05H

Instruction code : 

7	0
0 1 1 1	0 1 0 0

 Byte 1

7	0
0 0 0 0	0 1 0 1

 Byte 2

Before execution

After execution

Accumulator

Accumulator

0 1 1 1	0 1 1 1
7	0

0 0 0 0	0 1 0 1
7	0

**56. MOV A, @Rr (Move indirect address to accumulator)**

Instruction code	7	0
	1 1 1 0 0 1 1	r
		Byte 1

Operation	: $(A) \leftarrow ((Rr)) \quad r = 0 \text{ or } 1$
-----------	---

Number of bytes	: 1
-----------------	-----

Number of cycles	: 1
------------------	-----

Flags	C	AC	F0	RS1	RS0	OV	F1	P
(PSW)								

**Description**: The data memory location contents addressed by the register  $r$  contents are copied to the accumulator, and the flag is updated.

Example    MOV A, @R0

Instruction code : 

7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	0

 Byte 1

Before execution

After execution

Register 0

Register 0

0 1 1 1 0 0 1 0

0 1 1 1 0 0 1 0

7	0
---	---

---

7
0

72H

72H

1 0 1 1 0 1 1 1

1 0 1 1 0 1 1 1

7	0
---	---

7 0

## Accumulator

Accumulator

0 1 0 0 1 1 0 0

1 0 1 1 0 1 1 1

7 0

7 0

57. MOV A, Rr (Move register to accumulator)

Instruction code : 

7	0
1 1 1 0	1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>

 Byte 1

Operation : (A) ← (Rr) r = 0 ~ 7

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

•								•
---	--	--	--	--	--	--	--	---

Description : The register r contents are copied to the accumulator, and the flag is updated.

Example MOV A, R6

Instruction code : 

7	0
1 1 1 0 1 1 1 0	

 Byte 1

Before execution

After execution

Register 6

Register 6

7	0
1 0 1 0 0 1 0 1	

7	0
1 0 1 0 0 1 0 1	

Accumulator

Accumulator

7	0
0 0 0 0 1 1 1 1	

7	0
1 0 1 0 0 1 0 1	

58. MOV A, data address (Move memory to accumulator)

Instruction code : 

7	0
1	1
1	0
0	0
1	0
1	1

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub>
a <sub>5</sub>	a <sub>4</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (A) ← (data address)

Number of bytes : 2

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

.	.	.	.	.	.	.	.
---	---	---	---	---	---	---	---

Description : The specified data address contents are copied to the accumulator, and the flag is updated.

Example MOV A, P1

Instruction code : 

7	0
1	1
1	0
0	0
1	0
1	1

 Byte 1

7	0
1	0
0	0
1	0
0	0
0	0

 Byte 2

Before execution

After execution

Port 1

Port 1

0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

7 0

7 0

Accumulator

Accumulator

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

7 0

7 0

59. MOV C, bit address (Move bit to carry flag)

Instruction code : 

7	0
1	0
1	0
0	0
0	0
1	0
0	0

 Byte 1

Bit address : 

7	0
b <sub>7</sub>	b <sub>6</sub>
b <sub>5</sub>	b <sub>4</sub>
b <sub>3</sub>	b <sub>2</sub>
b <sub>1</sub>	b <sub>0</sub>

 Byte 2

Operation : (C) ← (bit address)

Number of bytes : 2

Number of cycles : 1

Flags : 

C	AC	F0	RS1	RS0	OV	F1	P
●							

  
(PSW)

Description : The specified bit address content is copied to the carry flag.

Example MOV C, P3.4

Instruction code : 

7	0
1	0
1	0
0	0
0	0
1	0
0	0

 Byte 1

7	0
1	0
1	1
0	1
0	0
0	0
0	0

 Byte 2

Before execution

Port 3

7	4	0
0	0	0
1	0	1
1	0	0

Carry flag

0
---

After execution

Port 3

7	4	0
0	0	0
1	0	1
1	0	0

Carry flag

1
---

60. MOV DPTR, #data (Move immediate data to data pointer)

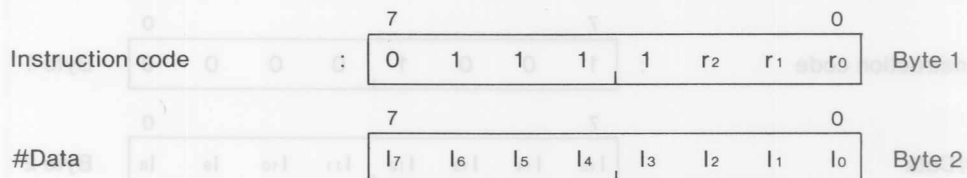
Instruction code	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	7							0	1	0	0	1	0	0	0	0	Byte 1
7							0												
1	0	0	1	0	0	0	0												
#Data	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>I<sub>15</sub></td><td>I<sub>14</sub></td><td>I<sub>13</sub></td><td>I<sub>12</sub></td><td>I<sub>11</sub></td><td>I<sub>10</sub></td><td>I<sub>9</sub></td><td>I<sub>8</sub></td></tr></table>	7							0	I <sub>15</sub>	I <sub>14</sub>	I <sub>13</sub>	I <sub>12</sub>	I <sub>11</sub>	I <sub>10</sub>	I <sub>9</sub>	I <sub>8</sub>	Byte 2
7							0												
I <sub>15</sub>	I <sub>14</sub>	I <sub>13</sub>	I <sub>12</sub>	I <sub>11</sub>	I <sub>10</sub>	I <sub>9</sub>	I <sub>8</sub>												
#Data	:	<table><tr><td>7</td><td colspan="6"></td><td>0</td></tr><tr><td>I<sub>7</sub></td><td>I<sub>6</sub></td><td>I<sub>5</sub></td><td>I<sub>4</sub></td><td>I<sub>3</sub></td><td>I<sub>2</sub></td><td>I<sub>1</sub></td><td>I<sub>0</sub></td></tr></table>	7							0	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>	Byte 3
7							0												
I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>												
Operation	:	(DPTR) ← #data (DPH) ← I <sub>8-15</sub> (DPL) ← I <sub>0-7</sub>																	
Number of bytes	:	3																	
Number of cycles	:	2																	
Flags	:	C AC F0 RS1 RS0 OV F1 P																	
(PSW)	:	<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>																	
Description	:	A 16-bit immediate data value is copied to the data pointer (DPH·DPL).																	

Example MOV DPTR, #0AF5H

Instruction code	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	1	0	0	1	0	0	0	0	Byte 1
7	6	5	4	3	2	1	0												
1	0	0	1	0	0	0	0												
	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	0	0	0	0	1	0	1	0	Byte 2
7	6	5	4	3	2	1	0												
0	0	0	0	1	0	1	0												
	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	7	6	5	4	3	2	1	0	1	1	1	1	0	1	0	1	Byte 3
7	6	5	4	3	2	1	0												
1	1	1	1	0	1	0	1												

Before execution				After execution			
DPH		DPL		DPH		DPL	
1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1
15	8	7	0	15	8	7	0

61. MOV Rr, #data (Move immediate data to register)



Operation : (Rr) ← #data r = 0 ~ 7

Number of bytes : 2

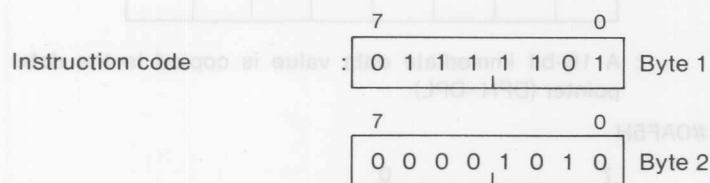
Number of cycles : 1

Flags : C AC FO RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : An 8-bit immediate data value is copied to the register r.

Example MOV R5, #0AH

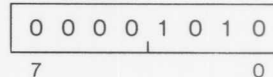
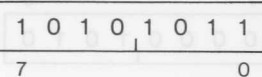


Before execution

After execution

Register 5

Register 5



62. MOV Rr, A (Move accumulator to register)

Instruction code : 

7	0
1	1
1	1
1	1
1	r <sub>2</sub>
r <sub>1</sub>	r <sub>0</sub>

 Byte 1

Operation : (Rr) ← (A) r = 0 ~ 7

Number of bytes : 1

Number of cycles : 1

Flags : C AC FO RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The accumulator contents are copied to the register r.

Example MOV R1, A

Instruction code : 

7	0
1	1
1	1
1	1
1	0
0	0
1	1

 Byte 1

Before execution

Register 1

0	1	1	1	1	1	1	0
---	---	---	---	---	---	---	---

Accumulator

1	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---

After execution

Register 1

1	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---

Accumulator

1	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---



63. MOV Rr, data address (Move memory to register)

Instruction code : 

7	0
1 0 1 0	1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>

 Byte 1

Data address : 

7	0
a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub>	a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>

 Byte 2

Operation : (Rr) ← (data address) r = 0 ~ 7

Number of bytes : 2

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The specified data address contents are copied to the register r.

Example MOV R0, 5AH

Instruction code : 

7	0
1 0 1 0	1 0 0 0

 Byte 1

7	0
0 1 0 1	1 0 1 0

 Byte 2

Before execution

Register 0

0	1	1	1	1	0	1	1
7							0

5AH

1	0	1	0	1	0	1	0
7							0

After execution

Register 0

1	0	1	0	1	0	1	0
7							0

5AH

1	0	1	0	1	0	1	0
7							0

64. MOV bit address, C (Move carry flag to bit)

Instruction code : 

7	0
1	0
0	1
0	0
1	0
0	1
0	0
0	0

 Byte 1

Bit address : 

7	0
b <sub>7</sub>	b <sub>0</sub>
b <sub>6</sub>	b <sub>5</sub>
b <sub>4</sub>	b <sub>3</sub>
b <sub>2</sub>	b <sub>1</sub>
b <sub>0</sub>	b <sub>7</sub>

 Byte 2

Operation : (bit address) ← (C)

Number of bytes : 2

Number of cycles : 2

Flags : C AC FO RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The carry flag content is copied to the specified bit address.

Example MOV P1.4, C

Instruction code : 

7	0
1	0
0	1
0	0
1	0
0	1
0	0
0	0

 Byte 1  

7	0
1	0
0	1
0	0
1	0
0	1
0	0
0	0

 Byte 2

Before execution

Port 1

7	4	0
1	1	1
1	1	1
1	1	1

Carry flag

0
---

After execution

Port 1

7	4	0
1	1	1
1	0	1
1	1	1

Carry flag

0
---

65. MOV data address, #data (Move immediate data to memory)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0</div><div>1</div><div>1</div><div>1</div> <div>0</div><div>1</div><div>0</div><div>1</div> </div>	Byte 1
Data address	:	<div> <div>7</div> <div>0</div> <div>a<sub>7</sub></div><div>a<sub>6</sub></div><div>a<sub>5</sub></div><div>a<sub>4</sub></div> <div>a<sub>3</sub></div><div>a<sub>2</sub></div><div>a<sub>1</sub></div><div>a<sub>0</sub></div> </div>	Byte 2
#Data	:	<div> <div>7</div> <div>0</div> <div>l<sub>7</sub></div><div>l<sub>6</sub></div><div>l<sub>5</sub></div><div>l<sub>4</sub></div> <div>l<sub>3</sub></div><div>l<sub>2</sub></div><div>l<sub>1</sub></div><div>l<sub>0</sub></div> </div>	Byte 3
Operation	:	(data address) ← #data	
Number of bytes	:	3	
Number of cycles	:	2	
Flags (PSW)	:	<div> <div>C</div><div>AC</div><div>F0</div><div>RS1</div><div>RS0</div><div>OV</div><div>F1</div><div>P</div> </div>	

Description : An 8-bit immediate data value is copied to the specified data address.

Example MOV TCON, #50H

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0</div><div>1</div><div>1</div><div>1</div> <div>0</div><div>1</div><div>0</div><div>1</div> </div>	Byte 1
	:	<div> <div>7</div> <div>0</div> <div>1</div><div>0</div><div>0</div><div>0</div> <div>1</div><div>0</div><div>0</div><div>0</div> </div>	Byte 2
	:	<div> <div>7</div> <div>0</div> <div>0</div><div>1</div><div>0</div><div>1</div> <div>0</div><div>0</div><div>0</div><div>0</div> </div>	Byte 3
Before execution			
TCON (88H)			
	:	<div> <div>7</div> <div>0</div> <div>0</div><div>0</div><div>0</div><div>0</div> <div>0</div><div>0</div><div>0</div><div>0</div> </div>	
After execution			
TCON (88H)			
	:	<div> <div>7</div> <div>0</div> <div>0</div><div>1</div><div>0</div><div>1</div> <div>0</div><div>0</div><div>0</div><div>0</div> </div>	

66. MOV data address, @Rr (Move indirect address to memory)

Instruction code : 

7	0
1	0 0 0 0 0 1 1 r

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>

 Byte 2

Operation : (data address)  $\leftarrow$  ((Rr)) r = 0 or 1

Number of bytes : 2

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The data memory location contents addressed by the register r contents are copied to the specified data address.

Example MOV ACC, @R1

Instruction code : 

7	0
1	0 0 0 0 0 1 1 1

 Byte 1  

7	0
1	1 1 0 0 0 0 0 0

 Byte 2

Before execution

Accumulator

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

7 0

Register 1

0	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---

7 0

25H

0	1	1	0	1	1	1	1
---	---	---	---	---	---	---	---

7 0

After execution

Accumulator

0	1	1	0	1	1	1	1
---	---	---	---	---	---	---	---

7 0

Register 1

0	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---

7 0

25H

0	1	1	0	1	1	1	1
---	---	---	---	---	---	---	---

7 0

7

67. MOV data address, A (Move accumulator to memory)

Instruction code : 

7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1

 Byte 1

Data address : 

7	6	5	4	3	2	1	0
a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (data address) ← (A)

Number of bytes : 2

Number of cycles : 1

Flags : C AC FO RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The accumulator contents are copied to the specified data address.

Example MOV P3, A

Instruction code : 

7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1

 Byte 1

7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0

 Byte 2

Before execution

After execution

Port 3

Port 3

7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0

Accumulator

Accumulator

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0

## 68. MOV data address, Rr (Move register to memory)

Instruction code : 

7	0
1	0
0	0
0	1
r <sub>2</sub>	r <sub>1</sub>
r <sub>0</sub>	r <sub>0</sub>

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub>
a <sub>5</sub>	a <sub>4</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (data address) ← (Rr) r = 0 ~ 7

Number of bytes : 2

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The register r contents are copied to the specified data address.

Example MOV 6BH, R2

Instruction code : 

7	0
1	0
0	0
0	1
0	0
1	0
1	0
0	0

 Byte 1

7	0
0	1
1	0
0	1
0	0
1	0
1	0
1	1

 Byte 2

Before execution

After execution

6BH

6BH

7	0
1	0
1	1
0	1
1	1
0	1
1	0
0	0

7	0
0	1
0	1
0	1
0	1
0	1
0	1
1	0

Register 2

Register 2

7	0
0	1
0	1
0	1
0	1
0	1
0	1
1	0

7	0
0	1
0	1
0	1
0	1
0	1
0	1
1	0

69. MOV data address 1, data address 2 (Move memory to memory)

Instruction code : 

7	0
1	0
0	0
0	0
0	1
0	1
0	1
1	1

 Byte 1

Data address 2 : 

7	0
$a^2_7$	$a^2_6$
$a^2_5$	$a^2_4$
$a^2_3$	$a^2_2$
$a^2_1$	$a^2_0$

 Byte 2

Data address 1 : 

7	0
$a^1_7$	$a^1_6$
$a^1_5$	$a^1_4$
$a^1_3$	$a^1_2$
$a^1_1$	$a^1_0$

 Byte 3

Operation : (data address 1)  $\leftarrow$  (data address 2)

Number of bytes : 3

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The source data address (data address 2) contents are copied to the destination data address (data address 1)

Example MOV ACC, P1

Instruction code : 

7	0
1	0
0	0
0	0
0	1
0	1
0	1
1	1

 Byte 1

7	0
1	0
0	1
0	0
0	0
0	0
0	0
0	0

 Byte 2

7	0
1	1
1	0
0	0
0	0
0	0
0	0
0	0

 Byte 3

Before execution

After execution

Port 1

Port 1

7	0
1	0
1	1
0	1
0	1
0	0
0	0
0	0

7	0
1	0
1	1
0	1
0	1
0	0
0	0
0	0

Accumulator

Accumulator

7	0
0	1
1	1
1	1
1	0
1	1
1	1
1	1

7	0
1	0
1	1
1	0
1	1
0	1
0	0
0	0

70. MOVC A, @A + DPTR

(Move code memory offset from data pointer to accumulator)

Instruction code : 

7	0
1	0
0	1
0	0
1	1

 Byte 1

Operation :  $(A) \leftarrow ((A) + (DPTR))$

Number of bytes : 1

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

							•
--	--	--	--	--	--	--	---

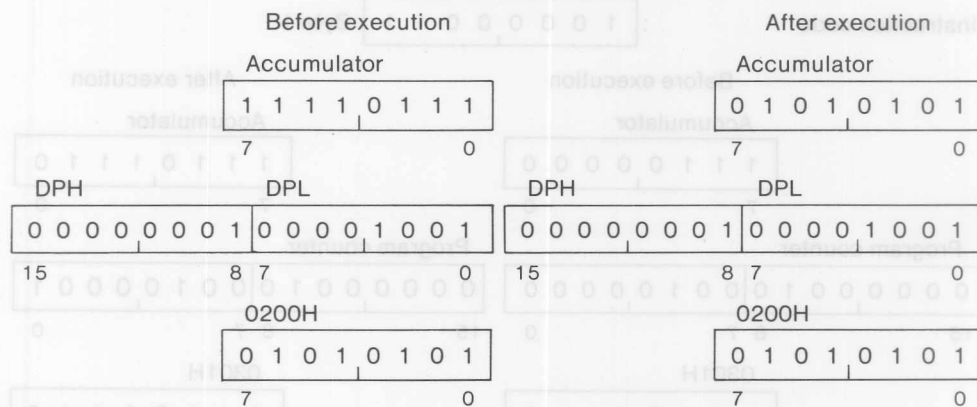
Description : The data pointer contents are added to the accumulator contents, and after temporary storage of the sum in program counter, the ROM data contents specified by the program counter are stored in the accumulator. The program counter contents are then restored to former contents, and the flag is also updated.

Example MOVC A, @A + DPTR

Instruction code : 

7	0
1	0
0	1
0	0
1	1

 Byte 1





**71. MOVC A, @A + PC**

(Move code memory offset from program counter to accumulator)

Instruction code : 

7	0
1	0
0	0
0	0
0	1
1	1

 Byte 1

Operation :  $(PC) \leftarrow (PC) + 1$   
 $(A) \leftarrow ((A) + (PC))$

Number of bytes : 1

Number of cycles : 2

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
							●

Description : The program counter contents following an increment are added to the accumulator contents, and after temporary storage of the sum in the program counter, the ROM data contents specified by the program counter are stored in the accumulator. The program counter contents are then restored to former contents, and the flag is also updated.

Example MOVC A, @A + PC

Instruction code : 

7	0
1	0
0	0
0	0
0	1
1	1

 Byte 1

Before execution				After execution																																			
Accumulator		<table border="1" style="width: 100%;"><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>		1	1	1	0	0	0	0	0	7							0	Accumulator		<table border="1" style="width: 100%;"><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>		1	1	1	0	1	1	1	0	7							0
1	1	1	0	0	0	0	0																																
7							0																																
1	1	1	0	1	1	1	0																																
7							0																																
Program counter		<table border="1" style="width: 100%;"><tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">15</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td colspan="4"></td><td style="text-align: center;">0</td></tr></table>		0	0	0	0	0	0	1	0	15	8	7					0	Program counter		<table border="1" style="width: 100%;"><tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">15</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td colspan="4"></td><td style="text-align: center;">0</td></tr></table>		0	0	0	0	0	0	1	0	15	8	7					0
0	0	0	0	0	0	1	0																																
15	8	7					0																																
0	0	0	0	0	0	1	0																																
15	8	7					0																																
0301H		<table border="1" style="width: 100%;"><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>		1	1	1	0	1	1	1	0	7							0	0301H		<table border="1" style="width: 100%;"><tr><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>		1	1	1	0	1	1	1	0	7							0
1	1	1	0	1	1	1	0																																
7							0																																
1	1	1	0	1	1	1	0																																
7							0																																

72. MOVX @DPTR, A

(Move accumulator to external memory addressed by data pointer)

Instruction code : 

7	0
1	0
1	0
1	0
1	0
0	0
0	0
0	0

 Byte 1

Operation : ((DPTR)) ← (A)

Number of bytes : 1

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--	--	--

Description : The accumulator contents are stored to external data memory (RAM) addressed by the data pointer contents.

Example MOVX @DPTR, A

Instruction code : 

7	0
1	1
1	1
1	0
0	0
0	0
0	0
0	0

 Byte 1

Before execution								After execution							
DPH				DPL				DPH				DPL			
0 1 1 0 0 0 1 0				1 1 0 0 1 1 0 0				0 1 1 0 0 0 1 0				1 1 0 0 1 1 0 0			
15				8 7				15				8 7			
62CCH								62CCH							
1 0 1 1 1 0 0 0								0 0 0 0 0 1 0 1							
7				0				7				0			
Accumulator								Accumulator							
0 0 0 0 0 1 0 1								0 0 0 0 0 1 0 1							
7				0				7				0			

### 73. MOVX @Rr, A

(Move accumulator to external memory addressed by register)

Instruction code: 

7	0
1	1
1	1
1	0
0	0
1	r

 Byte 1

Operation:  $((Rr)) \leftarrow (A)$   $r = 0 \text{ or } 1$

Number of bytes: 1

Number of cycles: 2

Flags: C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description: The accumulator contents are stored to external data memory (RAM) addressed by the register r contents.

Example MOVX @R0, A

Instruction code: 

7	0
1	1
1	1
1	0
0	0
1	0

 Byte 1

	Before execution	After execution																																
Register 0	<table><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr></table>	1	0	1	0	0	0	0	0	7							0	<table><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr></table>	1	0	1	0	0	0	0	0	7							0
1	0	1	0	0	0	0	0																											
7							0																											
1	0	1	0	0	0	0	0																											
7							0																											
0A0H	<table><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr></table>	0	0	1	1	0	0	1	1	7							0	<table><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr></table>	1	0	1	1	1	1	0	1	7							0
0	0	1	1	0	0	1	1																											
7							0																											
1	0	1	1	1	1	0	1																											
7							0																											
Accumulator	<table><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr></table>	1	0	1	1	1	1	0	1	7							0	<table><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr></table>	1	0	1	1	1	1	0	1	7							0
1	0	1	1	1	1	0	1																											
7							0																											
1	0	1	1	1	1	0	1																											
7							0																											

# 74. MOVX A, @DPTR

(Move external memory addressed by data pointer to accumulator)

Instruction code : 

7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	0

 Byte 1

Operation :  $(A) \leftarrow ((DPTR))$

Number of bytes : 1

Number of cycles : 2

Flags : 

C	AC	F0	RS1	RS0	OV	F1	P
							•

  
(PSW)

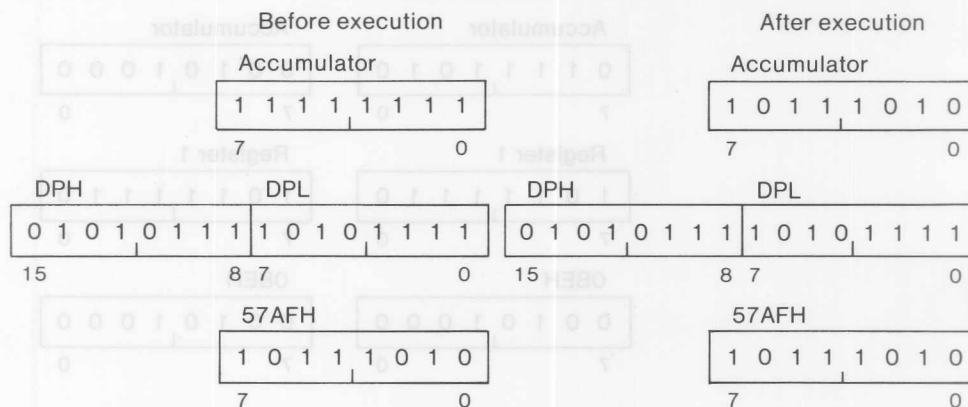
Description : External data memory (RAM) contents addressed by the data pointer are stored to the accumulator, and the flag is updated.

Example MOVX A, @DPTR

Instruction code : 

7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	0

 Byte 1



75. MOVX A, @Rr (Move external memory addressed by register to accumulator)

Instruction code : 

7	0
1	1
1	0
0	0
0	1
r	

 Byte 1

Operation :  $(A) \leftarrow ((Rr))$   $r = 0 \text{ or } 1$

Number of bytes : 1

Number of cycles : 2

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
							●

Description : External data memory (RAM) contents addressed by the register r contents are stored to the accumulator, and the flag is updated.

Example MOVX A, @R1

Instruction code : 

7	0
1	1
1	0
0	0
0	1
1	1

 Byte 1

Before execution	After execution																																
Accumulator	Accumulator																																
<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>	0	1	1	1	1	0	1	0	7							0	<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>	0	0	1	0	1	0	0	0	7							0
0	1	1	1	1	0	1	0																										
7							0																										
0	0	1	0	1	0	0	0																										
7							0																										
Register 1	Register 1																																
<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>	1	0	1	1	1	1	1	0	7							0	<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>	1	0	1	1	1	1	1	0	7							0
1	0	1	1	1	1	1	0																										
7							0																										
1	0	1	1	1	1	1	0																										
7							0																										
OBEH	OBEH																																
<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>	0	0	1	0	1	0	0	0	7							0	<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">7</td><td colspan="6"></td><td style="text-align: center;">0</td></tr></table>	0	0	1	0	1	0	0	0	7							0
0	0	1	0	1	0	0	0																										
7							0																										
0	0	1	0	1	0	0	0																										
7							0																										

76. MUL AB (Multiply accumulator by B)

Instruction code : 

7	0
1 0 1 0	0 1 0 0

 Byte 1

Operation :  $(A)_{0-7} \leftarrow (A) \times (B)$   
(B)<sub>8-15</sub>

Number of bytes : 1

Number of cycles : 4

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
•					•		•

Description : The accumulator contents are multiplied by the arithmetic operation register (B) contents. The operand is always handled as an integer without sign. The lower order byte of the result is stored in the accumulator, and the higher order byte is stored in the arithmetic operation register (B). The carry flag is always cleared. The overflow flag is set to 1 if the product is greater than 00FFH, and to 0 in all other cases.

Example MUL AB (6AH × 15H = 8B2H)

Instruction code : 

7	0
1 0 1 0	0 1 0 0

 Byte 1

Before execution

Accumulator

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

7 0

Register B

0	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---

7 0

Overflow flag

0
---

After execution

Accumulator

1	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

7 0

Register B

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

7 0

Overflow flag

1
---

## 77. NOP (No operation)

Instruction code : 

7	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

 Byte 1

Operation :  $(PC) \leftarrow (PC) + 1$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P

(PSW) 

--	--	--	--	--	--	--	--

Description : The program counter is incremented by 1 without any other change in the CPU. Control is shifted to the next instruction.

78. ORL A, #data (Logical OR immediate data to accumulator)

Instruction code : 

7	0
0	1
0	0
0	0
0	1
0	0
0	0
0	0

 Byte 1

#data : 

7	0
I <sub>7</sub>	I <sub>6</sub>
I <sub>5</sub>	I <sub>4</sub>
I <sub>3</sub>	I <sub>2</sub>
I <sub>1</sub>	I <sub>0</sub>

 Byte 2

Operation : (A) ← (A) OR #data

Number of bytes : 2

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

							•
--	--	--	--	--	--	--	---

Description : The logical OR between an 8-bit immediate data value and the accumulator contents is determined. The result is stored in the accumulator and the flag is also updated.

Example ORL A, #5FH

Instruction code : 

7	0
0	1
0	0
0	0
0	1
0	0
0	0
0	0

 Byte 1  

7	0
0	1
0	1
1	1
1	1
1	1
1	1
1	1

 Byte 2

Before execution

Accumulator

7	0
1	0
0	0
0	0
0	1
0	0
0	0
0	0

After execution

Accumulator

7	0
1	1
0	1
1	1
1	1
1	1
1	1
1	1



79. ORL A, @Rr (Logical OR indirect address to accumulator)

Instruction code : 

7	0
0	1
0	0
0	1
0	1
1	r

 Byte 1

Operation :  $(A) \leftarrow (A) \text{ OR } ((Rr))$   $r = 0 \text{ or } 1$

Number of bytes : 1

Number of cycles : 1

Flags : C AC FO RS1 RS0 OV F1 P  
(PSW) 

							●
--	--	--	--	--	--	--	---

Description : The logical OR between the accumulator contents and the data memory location contents addressed by the register r contents is determined. The result is stored in the accumulator and the flag is also updated.

Example ORL A, @R0

Instruction code : 

7	0
0	1
0	0
0	1
1	1
0	0

 Byte 1

Before execution

Accumulator

0	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---

7 0

Register 0

0	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

7 0

6DH

1	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---

7 0

After execution

Accumulator

1	0	1	1	1	1	1	1
---	---	---	---	---	---	---	---

7 0

Register 0

0	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

7 0

6DH

1	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---

7 0

80. ORL A, Rr (Logical OR register to accumulator)

Instruction code : 

7	0
0	1
0	0
1	r <sub>2</sub>
r <sub>1</sub>	r <sub>0</sub>

 Byte 1

Operation : (A) ← (A) OR (Rr) r = 0 ~ 7

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

							•
--	--	--	--	--	--	--	---

Description : The logical OR between the accumulator contents and the register r contents is determined. The result is stored in the accumulator and the flag is also updated.

Example ORL A, R5

Instruction code : 

7	0
0	1
0	0
1	1
0	1

 Byte 1

Before execution

Accumulator

0	0	0	0	0	0	0	0
7							0

Register 5

0	1	1	1	0	1	0	1
7							0

After execution

Accumulator

0	1	1	1	0	1	0	1
7							0

Register 5

0	1	1	1	0	1	0	1
7							0

**81. ORL A, data address (Logical OR memory to accumulator)**

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0 1 0 0 0 1 0 1</div> </div>	Byte 1
Data address	:	<div> <div>7</div> <div>0</div> <div>a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub></div> </div>	Byte 2
Operation	:	(A) ← (A) OR (data address)	
Number of bytes	:	2	
Number of cycles	:	1	
Flags	:	<div> <div>C</div> <div>AC</div> <div>F0</div> <div>RS1</div> <div>RS0</div> <div>OV</div> <div>F1</div> <div>P</div> </div>	
(PSW)	:	<div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div>•</div> </div>	

**Description** : The logical OR between the accumulator contents and the specified data address contents is determined. The result is stored in the accumulator and the flag is also updated.

**Example** ORL A, 33H

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0 1 0 0 0 1 0 1</div> </div>	Byte 1
	:	<div> <div>7</div> <div>0</div> <div>0 0 1 1 0 0 1 1</div> </div>	Byte 2
<div> <div>Before execution</div> <div>Accumulator</div> <div>0 1 0 1 1 1 1 0</div> <div>7 0</div> <div>33H</div> <div>1 0 1 0 0 1 0 1</div> <div>7 0</div> </div>			
<div> <div>After execution</div> <div>Accumulator</div> <div>1 1 1 1 1 1 1 1</div> <div>7 0</div> <div>33H</div> <div>1 0 1 0 0 1 0 1</div> <div>7 0</div> </div>			

82. ORL C, bit address (Logical OR bit to carry flag)

Instruction code : 

7	0
0	1
1	1
1	0
0	0
1	0

 Byte 1

Bit address : 

7	0
b <sub>7</sub>	b <sub>6</sub>
b <sub>5</sub>	b <sub>4</sub>
b <sub>3</sub>	b <sub>2</sub>
b <sub>1</sub>	b <sub>0</sub>

 Byte 2

Operation : (C) ← (C) OR (bit address)

Number of bytes : 2

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

•							
---	--	--	--	--	--	--	--

Description : The logical OR between the carry flag and the specified bit address content is determined. The result is stored in the carry flag.

Example ORL C, ACC. 6

Instruction code : 

7	0
0	1
1	1
1	0
0	0
1	0

 Byte 1

Byte 2 : 

7	0
1	1
1	0
0	0
1	1
1	0

 Byte 2

After execution

Before execution

After execution

Carry flag

Carry flag

0

1

Accumulator

Accumulator

7	6	0
0	1	1
1	1	0
0	0	1
0	1	0

7	6	0
0	1	1
1	1	0
0	0	1
0	1	0

7

83. ORL C,/bit address (Logical OR complement of bit to carry flag)

Instruction code : 

7	0
1	0
1	0
0	0
0	0
0	0
0	0
0	0

 Byte 1

Bit address : 

7	0
b <sub>7</sub>	b <sub>0</sub>
b <sub>6</sub>	b <sub>1</sub>
b <sub>5</sub>	b <sub>2</sub>
b <sub>4</sub>	b <sub>3</sub>
b <sub>3</sub>	b <sub>2</sub>
b <sub>2</sub>	b <sub>1</sub>
b <sub>1</sub>	b <sub>0</sub>

 Byte 2

Operation :  $(C) \leftarrow (C) \text{ OR } (\text{bit address})$

Number of bytes : 2

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

•								
---	--	--	--	--	--	--	--	--

Description : The logical OR between the carry flag and the complement of specified bit address content is determined. The result is stored in the carry flag.

Example ORL C,/25H. 5

Instruction code : 

7	0
1	0
1	0
0	0
0	0
0	0
0	0
0	0

 Byte 1

Byte 2 : 

7	0
0	0
1	0
1	1
0	1
1	0
1	0
1	0

 Byte 2

After execution

Before execution

After execution

Carry flag

Carry flag

0

1

25H

25H

7	5	0
1	0	0
0	1	0
1	0	1
0	1	0

7	5	0
1	0	0
0	1	0
1	0	1
0	1	0

**84. ORL data address, #data (Logical OR immediate data to memory)**

Instruction code	:	<table><tr><td>7</td><td>0</td></tr><tr><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	7	0	0	1	0	0	0	0	1	1	Byte 1
7	0												
0	1												
0	0												
0	0												
1	1												
Data address	:	<table><tr><td>7</td><td>0</td></tr><tr><td>a<sub>7</sub></td><td>a<sub>6</sub></td></tr><tr><td>a<sub>5</sub></td><td>a<sub>4</sub></td></tr><tr><td>a<sub>3</sub></td><td>a<sub>2</sub></td></tr><tr><td>a<sub>1</sub></td><td>a<sub>0</sub></td></tr></table>	7	0	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2
7	0												
a <sub>7</sub>	a <sub>6</sub>												
a <sub>5</sub>	a <sub>4</sub>												
a <sub>3</sub>	a <sub>2</sub>												
a <sub>1</sub>	a <sub>0</sub>												
#data	:	<table><tr><td>7</td><td>0</td></tr><tr><td>l<sub>7</sub></td><td>l<sub>6</sub></td></tr><tr><td>l<sub>5</sub></td><td>l<sub>4</sub></td></tr><tr><td>l<sub>3</sub></td><td>l<sub>2</sub></td></tr><tr><td>l<sub>1</sub></td><td>l<sub>0</sub></td></tr></table>	7	0	l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>	Byte 3
7	0												
l <sub>7</sub>	l <sub>6</sub>												
l <sub>5</sub>	l <sub>4</sub>												
l <sub>3</sub>	l <sub>2</sub>												
l <sub>1</sub>	l <sub>0</sub>												
Operation	:	(data address) ← (data address) OR #data											
Number of bytes	:	3											
Number of cycles	:	2											
Flags	:	C AC F0 RS1 RS0 OV F1 P											
(PSW)	:	<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>											

**Description** : The logical OR between an 8-bit immediate data value and the specified data address contents is determined. The result is stored in the specified data address.

**Example** ORL 55H, #11H

Instruction code	:	<table><tr><td>7</td><td>0</td></tr><tr><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	7	0	0	1	0	0	0	0	1	1	Byte 1												
7	0																								
0	1																								
0	0																								
0	0																								
1	1																								
	:	<table><tr><td>7</td><td>0</td></tr><tr><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td></tr></table>	7	0	0	1	0	1	0	1	0	1	Byte 2												
7	0																								
0	1																								
0	1																								
0	1																								
0	1																								
	:	<table><tr><td>7</td><td>0</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table>	7	0	0	0	0	1	0	0	0	1	Byte 3												
7	0																								
0	0																								
0	1																								
0	0																								
0	1																								
	:	Before execution	After execution																						
	:	<table><tr><td>55H</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>7</td><td>0</td></tr></table>	55H	1	0	0	0	0	1	0	0	7	0	<table><tr><td>55H</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>7</td><td>0</td></tr></table>	55H	1	0	0	1	0	1	0	1	7	0
55H																									
1	0	0	0	0	1	0	0																		
7	0																								
55H																									
1	0	0	1	0	1	0	1																		
7	0																								

85. ORL data address, A (Logical OR accumulator to memory)

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0 1 0 0 0 0 1 0</div> </div>	Byte 1
Data address	:	<div> <div>7</div> <div>0</div> <div>a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub></div> </div>	Byte 2
Operation	:	(data address) ← (data address) OR (A)	
Number of bytes	:	2	
Number of cycles	:	1	
Flags (PSW)	:	<div> <div>C</div> <div>AC</div> <div>F0</div> <div>RS1</div> <div>RS0</div> <div>OV</div> <div>F1</div> <div>P</div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>	
Description	:	The logical OR between the accumulator and the specified data address contents is determined. The result is stored in the specified data address.	

Example ORL 50H, A

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0 1 0 0 0 0 1 0</div> </div>	Byte 1
	:	<div> <div>7</div> <div>0</div> <div>0 1 0 1 0 0 0 0</div> </div>	Byte 2
Before execution			
Accumulator		<div> <div>7</div> <div>0</div> <div>1 0 0 0 1 1 1 1</div> </div>	
50H		<div> <div>7</div> <div>0</div> <div>0 0 1 0 0 1 0 1</div> </div>	
After execution			
Accumulator		<div> <div>7</div> <div>0</div> <div>1 0 0 0 1 1 1 1</div> </div>	
50H		<div> <div>7</div> <div>0</div> <div>1 0 1 0 1 1 1 1</div> </div>	

86. POP data address (Pop stack to memory)

Instruction code : 

7	0
1	0
1	0
0	0
1	0
0	0
0	0
0	0

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>0</sub>
a <sub>6</sub>	a <sub>1</sub>
a <sub>5</sub>	a <sub>2</sub>
a <sub>4</sub>	a <sub>3</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>2</sub>	a <sub>1</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (data address)  $\leftarrow$  ((SP))  
(SP)  $\leftarrow$  (SP) - 1

Number of bytes : 2

Number of cycles : 2

Flags : C AC FO RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : Stack contents addressed by the stack pointer are transferred to the specified data address, and the stack pointer is decremented by 1.

Example POP PSW: No change to parity bit.

Instruction code : 

7	0
1	0
1	0
0	0
1	0
0	0
0	0
0	0

 Byte 1

7	0
1	0
1	0
0	0
1	0
0	0
0	0
0	0

 Byte 2

Before execution

After execution

Accumulator

Accumulator

7	0
1	0
0	1
1	1
0	0
1	0
0	0
0	0

7	0
1	0
0	1
1	1
0	0
1	0
0	0
0	0

7 0

7 0

PSW (0D0H)

PSW (0D0H)

7	0
1	0
0	1
1	1
0	0
1	0
0	0
0	0

7	0
1	1
1	1
1	0
0	0
1	0
0	1
0	0

7 0

7 0

Stack pointer

Stack pointer

7	0
0	0
0	1
0	0
0	0
0	0
0	0
0	0

7	0
0	0
0	0
0	1
1	1
1	1
1	1
1	1

7 0

7 0

10H

10H

7	0
1	1
1	1
1	0
0	0
1	1
1	1
1	1

7	0
1	1
1	1
1	0
0	0
1	1
1	1
1	1

7 0

7 0



87. PUSH data address (Push memory onto stack)

Instruction code	:	<table><tr><td>7</td><td colspan="7"></td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	7								0	1	1	0	0	0	0	0	0	Byte 1
7								0												
1	1	0	0	0	0	0	0													
Data address	:	<table><tr><td>7</td><td colspan="7"></td><td>0</td></tr><tr><td>a<sub>7</sub></td><td>a<sub>6</sub></td><td>a<sub>5</sub></td><td>a<sub>4</sub></td><td>a<sub>3</sub></td><td>a<sub>2</sub></td><td>a<sub>1</sub></td><td>a<sub>0</sub></td></tr></table>	7								0	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2
7								0												
a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>													
Operation	:	$(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (\text{data address})$																		
Number of bytes	:	2																		
Number of cycles	:	2																		
Flags	:	C	AC	F0	RS1	RS0	OV	F1	P											
(PSW)	:																			

Description : The stack pointer is incremented by 1, and the specified data address contents are stored in the stack addressed by the stack pointer.

Example PUSH P1

Instruction code	:	<div> <div>7</div> <div>0</div> <div>1 1 0 0 0 0 0 0</div> </div>	Byte 1
	:	<div> <div>7</div> <div>0</div> <div>1 0 0 1 0 0 0 0</div> </div>	Byte 2
		Before execution	After execution
		Port 1 (90H)	Port 1 (90H)
		<div> <div>7</div> <div>0</div> <div>1 1 0 1 0 1 0 1</div> </div>	<div> <div>7</div> <div>0</div> <div>1 1 0 1 0 1 0 1</div> </div>
		Stack pointer	Stack pointer
		<div> <div>7</div> <div>0</div> <div>0 0 0 1 0 0 0 0</div> </div>	<div> <div>7</div> <div>0</div> <div>0 0 0 1 0 0 0 1</div> </div>
		11H (Stack)	11H (Stack)
		<div> <div>7</div> <div>0</div> <div>0 0 0 0 0 0 0 0</div> </div>	<div> <div>7</div> <div>0</div> <div>1 1 0 1 0 1 0 1</div> </div>

88. RET (Return from subroutine, non interrupt)

Instruction code : 

7	0
0	0
1	0
0	0
0	1
0	0

 Byte 1

Operation :  $(PC_{8-15}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$   
 $(PC_{0-7}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$

Number of bytes : 1

Number of cycles : 2

Flags : C AC F0 RS1 RS0 OV F1 P

(PSW)

--	--	--	--	--	--	--	--	--	--

Description : The stack contents addressed by the stack pointer are stored to upper order 8 thru 15 of the program counter, and the stack pointer is decremented by 1. Then the stack contents addressed by the updated stack pointer are stored in the lower order 0 thru 7 of the program counter, again decrementing the stack pointer by 1. The program counter is updated with the stack contents, and control is shifted to the address after updating.

## 89. RETI (Return from interrupt routine)

Instruction code : 

0	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

 Byte 1

Operation :  $(PC_{8-15}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$   
 $(PC_{0-7}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$   
 \*INTERRUPT ENABLE

Number of bytes : 1

Number of cycles : 2

Flags : C AC FO RS1 RS0 OV F1 P  
 (PSW) 

--	--	--	--	--	--	--	--

Description : This return instruction functions as an interrupt processing routine terminating instruction. If a priority interrupt is generated while a non-priority interrupt processing routine is being executed, the CPU starts to process the priority intrerupt. And once processing of this interrupt is started, no other interrupt can be processed until the RETI instruction is executed.

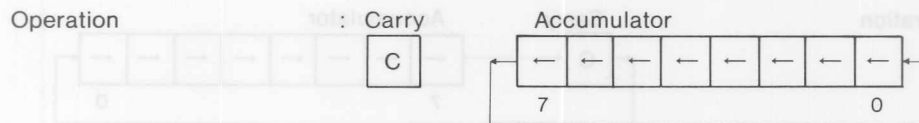
The stack contents addressed by the stack pointer are stored to upper order 8 thru 15 of the program counter, and the stack pointer is decremented by 1. Then the stack contents addressed by the updated stack pointer are stored in the lower order 0 thru 7 of the program counter, again decrementing the stack pointer by 1. The program counter is updated with the stack contents, and control is shifted to the address after updating. If a new interrupt is generated, the CPU starts to process the interrupt.

90. RL A (Rotate accumulator left)

Instruction code : 

7	0
0	0
1	0
0	0
0	1
1	1

 Byte 1



Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P

(PSW) 

--	--	--	--	--	--	--	--

Description : All accumulator bits are shifted by one bit to the left. The MSB (bit 7) is shifted to the LSB bit position (bit 0).

Example RL A

Instruction code : 

7	0
0	0
1	0
0	0
0	1
1	1

 Byte 1

Before execution

After execution

Accumulator

Accumulator

7	0
1	0
0	0
1	0
0	1
1	1
0	0

7	0
0	0
1	0
0	1
1	1
0	1

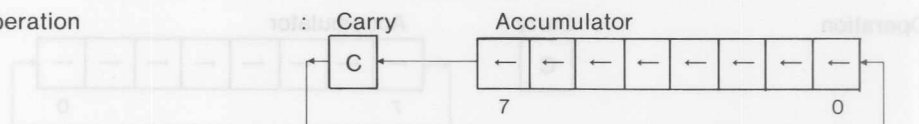
## 91. RLC A (Rotate accumulator and carry flag left)

Instruction code : 

7							0
0	0	1	1	0	0	1	1

 Byte 1

Operation



Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

•							•
---	--	--	--	--	--	--	---

Description : The accumulator and the carry flag are connected, and all bits are shifted by one bit to the left. The carry flag is shifted to the accumulator LSB (bit 0), and the accumulator MSB (bit 7) is shifted to the carry flag.

Example RLC A

Instruction code : 

7							0
0	0	1	1	0	0	1	1

 Byte 1

Before execution

Accumulator

0	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---

Carry flag

1
---

After execution

Accumulator

1	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---

Carry flag

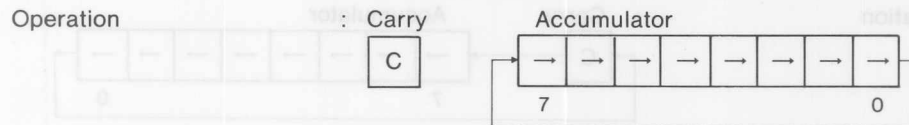
0
---

92. RR A (Rotate accumulator right)

Instruction code : 

7	0
0	0
0	0
0	0
0	1
0	1

 Byte 1



Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P

(PSW) 

--	--	--	--	--	--	--	--

Description : All accumulator bits are shifted by one bit to the right. The LSB (bit 0) is shifted to the MSB bit position (bit 7).

Example RR A

Instruction code : 

7	0
0	0
0	0
0	0
0	0
1	1

 Byte 1

Before execution

Accumulator

0	1	1	1	0	0	1	1
7							0

After execution

Accumulator

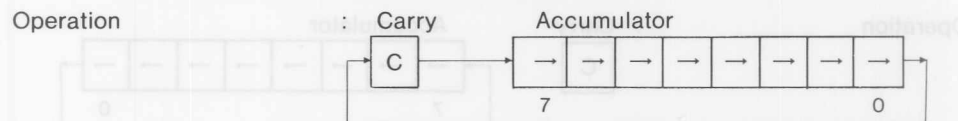
1	0	1	1	1	0	0	1
7							0

## 93. RRC A (Rotate accumulator and carry flag right)

Instruction code : 

7	0
0	0
0	1
0	0
1	1

 Byte 1



Number of bytes : 1

Number of cycles : 1

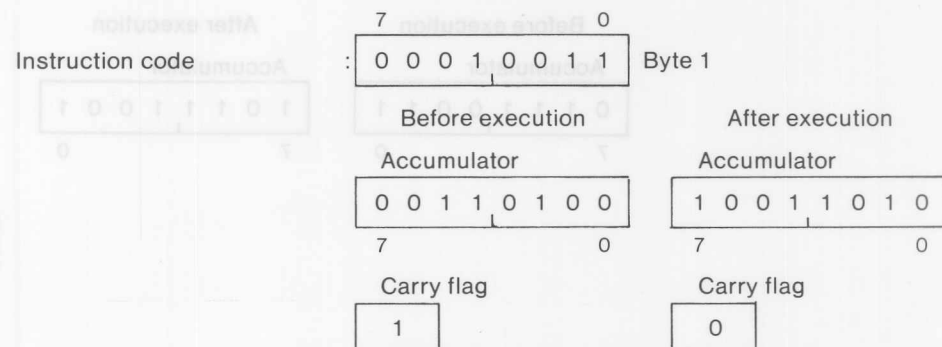
Flags : 

C	AC	F0	RS1	RS0	OV	F1	P
●							●

  
(PSW)

Description : The accumulator and the carry flag are connected, and all bits are shifted by one bit to the right. The carry flag is shifted to the accumulator MSB (bit 7), and the accumulator LSB (bit 0) is shifted to the carry flag.

Example RRC A



94. SETB C (Set carry flag)

Instruction code : 

7	0
1	1
0	1
0	0
1	1

 Byte 1

Operation :  $(C) \leftarrow 1$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

•							
---	--	--	--	--	--	--	--

Description : The carry flag is set to 1.

Example SETB C 

--	--	--	--	--	--

Instruction code : 

7	0
1	1
0	1
0	0
1	1

 Byte 1

Before execution

After execution

Carry flag

Carry flag

0

1



95. SETB bit address (Set bit)

Instruction code : 

7	0
1	1
0	1
0	0
1	0
0	1
0	0
0	0

 Byte 1

Bit address : 

7	0
b <sub>7</sub>	b <sub>6</sub>
b <sub>5</sub>	b <sub>4</sub>
b <sub>3</sub>	b <sub>2</sub>
b <sub>1</sub>	b <sub>0</sub>

 Byte 2

Operation : (bit address) ← 1

Number of bytes : 2

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

--	--	--	--	--	--	--	--

Description : The specified bit address content is set to 1.

Example SETB IE.7

Instruction code : 

7	0
1	1
0	1
0	0
1	0
0	1
0	0
0	0

 Byte 1

7	0
1	0
0	1
0	1
1	1
1	1
1	1
1	1

 Byte 2

Before execution

IE (A8H)

7	0
0	1
1	1
0	0
0	1
1	1
1	1
1	1

After execution

IE (A8H)

7	0
1	1
1	1
0	0
0	1
1	1
1	1
1	1

96. SJMP code address (Short jump)

Instruction code	:	<div><div>70</div><div>10000000</div></div> Byte 1
Relative offset	:	<div><div>70</div><div>R7R6R5R4R3R2R1R0</div></div> Byte 2
Operation	:	(PC) ← (PC) + 2 (PC) ← (PC) + relative offset
Number of bytes	:	2
Number of cycles	:	2
Flags	:	<div><div>CACF0RS1RS0OVF1P</div><div>00000000</div></div>

Description : Relative offset jump data is added/subtracted to/from the program counter contents following an increment. The program counter contents are updated, and control is then shifted to the updated address. The range in which relative jumps can be executed by this instruction is +127 to -128 in respect to the incremented program counter contents. There is no page field restrictions.

Example SJMP CHECK

LOC	OBJ	SOURCE
0111	8010	SJUMP : SJMP CHECK
0123	33	CHECK : RLC A

Instruction code

7	0	1	0	0	0	0	0	0	0	Byte 1
7	0	0	0	0	1	0	0	0	0	Byte 2

Before execution	After execution																																																																
Program counter	Program counter																																																																
<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td> <td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td> </tr> <tr> <td>15</td><td></td><td></td><td></td><td></td><td></td><td>8</td><td>7</td> <td></td><td></td><td></td><td></td><td>0</td><td></td><td></td><td></td> </tr> </table>	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	15						8	7					0				<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td> <td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td> </tr> <tr> <td>15</td><td></td><td></td><td></td><td></td><td></td><td>8</td><td>7</td> <td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td> </tr> </table>	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	15						8	7								0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1																																																		
15						8	7					0																																																					
0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1																																																		
15						8	7								0																																																		

97. **SUBB A, #data** (Subtract immediate data from accumulator with borrow)

Instruction code	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	1	0	0	1	0	1	0	0	Byte 1
7	6	5	4	3	2	1	0												
1	0	0	1	0	1	0	0												
#Data	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>I<sub>7</sub></td><td>I<sub>6</sub></td><td>I<sub>5</sub></td><td>I<sub>4</sub></td><td>I<sub>3</sub></td><td>I<sub>2</sub></td><td>I<sub>1</sub></td><td>I<sub>0</sub></td></tr></table>	7	6	5	4	3	2	1	0	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>	Byte 2
7	6	5	4	3	2	1	0												
I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>												
Operation	:	(A) ← (A) − ((C) + #data)																	
Number of bytes	:	2																	
Number of cycles	:	1																	
Flags	:	C	AC	F0	RS1	RS0	OV	F1	P										
(PSW)		●	●				●		●										

Description : The carry flag contents and an immediate data value are subtracted from the accumulator contents. The result is placed in the accumulator, and the flags are updated.

Example SUBB A, #05H

Instruction code	:	<div> <div>7</div> <div>0</div> <div>1 0 0 1 0 1 0 0</div> </div>	Byte 1
	:	<div> <div>7</div> <div>0</div> <div>0 0 0 0 0 1 0 1</div> </div>	Byte 2
Before execution			
Carry flag		1	
Auxiliary carry flag		0	
Overflow flag		1	
Accumulator		1 0 1 0 1 0 0 1	
After execution			
Carry flag		0	
Auxiliary carry flag		0	
Overflow flag		0	
Accumulator		1 0 1 0 0 0 1 1	

## 98. SUBB A, @Rr (Subtract indirect address from accumulator with borrow)

Instruction code	:	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"> <div style="display: flex; justify-content: space-between; margin: 0 5px;">70</div> <div style="display: flex; justify-content: space-between;"> <span>1</span><span>0</span><span>0</span><span>1</span><span>0</span><span>1</span><span>1</span><span>r</span> </div> </div> <span>Byte 1</span> </div>
Operation	:	$(A) \leftarrow (A) - ((C) + ((Rr)))$ $r = 0 \text{ or } 1$
Number of bytes	:	1
Number of cycles	:	1
Flags	:	C AC F0 RS1 RS0 OV F1 P
(PSW)	:	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;">•</div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;">•</div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"></div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"></div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"></div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;">•</div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"></div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;">•</div> </div>

Description : The carry flag contents, and the data memory location contents addressed by the register r contents are subtracted from the accumulator contents. The result is placed in the accumulator, and the flags are updated.

Example SUBB A, @R0

Instruction code	:	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"> <div style="display: flex; justify-content: space-between; margin: 0 5px;">70</div> <div style="display: flex; justify-content: space-between;"> <span>1</span><span>0</span><span>0</span><span>1</span><span>0</span><span>1</span><span>1</span><span>0</span> </div> </div> <span>Byte 1</span> </div>
Before execution		
Carry flag		<div style="border: 1px solid black; padding: 2px; text-align: center;">0</div>
Auxiliary carry flag		<div style="border: 1px solid black; padding: 2px; text-align: center;">0</div>
Overflow flag		<div style="border: 1px solid black; padding: 2px; text-align: center;">0</div>
Register 0		<div style="border: 1px solid black; padding: 2px; margin: 0 5px;"> <div style="display: flex; justify-content: space-between; margin: 0 5px;">70</div> <div style="display: flex; justify-content: space-between;"> <span>0</span><span>1</span><span>0</span><span>0</span><span>0</span><span>1</span><span>1</span><span>1</span> </div> </div> <div style="text-align: center;">47H</div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"> <div style="display: flex; justify-content: space-between; margin: 0 5px;">70</div> <div style="display: flex; justify-content: space-between;"> <span>1</span><span>1</span><span>0</span><span>1</span><span>0</span><span>0</span><span>1</span><span>0</span> </div> </div> <div style="text-align: center;">Accumulator</div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"> <div style="display: flex; justify-content: space-between; margin: 0 5px;">70</div> <div style="display: flex; justify-content: space-between;"> <span>0</span><span>1</span><span>0</span><span>1</span><span>0</span><span>1</span><span>0</span><span>0</span> </div> </div>
After execution		
Carry flag		<div style="border: 1px solid black; padding: 2px; text-align: center;">1</div>
Auxiliary carry flag		<div style="border: 1px solid black; padding: 2px; text-align: center;">0</div>
Overflow flag		<div style="border: 1px solid black; padding: 2px; text-align: center;">1</div>
Register 0		<div style="border: 1px solid black; padding: 2px; margin: 0 5px;"> <div style="display: flex; justify-content: space-between; margin: 0 5px;">70</div> <div style="display: flex; justify-content: space-between;"> <span>0</span><span>1</span><span>0</span><span>0</span><span>0</span><span>1</span><span>1</span><span>1</span> </div> </div> <div style="text-align: center;">47H</div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"> <div style="display: flex; justify-content: space-between; margin: 0 5px;">70</div> <div style="display: flex; justify-content: space-between;"> <span>1</span><span>1</span><span>0</span><span>1</span><span>0</span><span>0</span><span>1</span><span>0</span> </div> </div> <div style="text-align: center;">Accumulator</div> <div style="border: 1px solid black; padding: 2px; margin: 0 5px;"> <div style="display: flex; justify-content: space-between; margin: 0 5px;">70</div> <div style="display: flex; justify-content: space-between;"> <span>1</span><span>0</span><span>0</span><span>0</span><span>0</span><span>0</span><span>1</span><span>0</span> </div> </div>

## 99. SUBB A, Rr (Subtract register from accumulator with borrow)

Instruction code : 

7	0
1	0
0	1
1	1
r <sub>2</sub>	r <sub>1</sub>
r <sub>0</sub>	r <sub>0</sub>

 Byte 1

Operation :  $(A) \leftarrow (A) - ((C) + (Rr))$

Number of bytes : 1

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
•	•				•		•

Description : The carry flag contents and the register r contents are subtracted from the accumulator contents. The result is placed in the accumulator, and the flags are updated.

Example SUBB A, R7

Instruction code : 

7	0
1	0
0	1
1	1
1	1
1	1

 Byte 1

Before execution

Carry flag

1

Auxiliary carry flag

0

Overflow flag

0

Register 7

0	1	0	1	1	0	0	0
7							0

Accumulator

1	0	0	0	0	1	0	0
7							0

After execution

Carry flag

0

Auxiliary carry flag

1

Overflow flag

1

Register 7

0	1	0	1	1	0	0	0
7							0

Accumulator

0	0	1	0	1	0	1	1
7							0

7

100. SUBB A, data address (Subtract memory from accumulator with borrow)

Instruction code : 

7	0
1	0
0	1
0	1

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub>
a <sub>5</sub>	a <sub>4</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation :  $(A) \leftarrow (A) - ((C) + (\text{data address}))$

Number of bytes : 2

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
•	•				•		•

Description : The carry flag contents and the specified data address contents are subtracted from the accumulator contents. The result is placed in the accumulator, and the flags are updated.

Example SUBB A, DPH

Instruction code : 

7	0
1	0
0	1
0	1

 Byte 1

7	0
1	0
0	0
0	1
1	1

 Byte 2

Before execution

After execution

Carry flag

Carry flag

0
---

1
---

Auxiliary carry flag

Auxiliary carry flag

0
---

1
---

Overflow flag

Overflow flag

0
---

0
---

DPH

DPH

7	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0

7	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0

Accumulator

Accumulator

7	0
0	1
0	1
0	1
0	1
0	1
0	1
0	1

7	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0

**101. SWAP A (Exchange nibble in accumulator)**

Instruction code : 

7	0
1	0
1	0
0	0
0	1
0	0
0	0

 Byte 1

Operation :  $(A_{4-7}) \leftrightarrow (A_{0-3})$

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P

(PSW) 

--	--	--	--	--	--	--	--	--	--

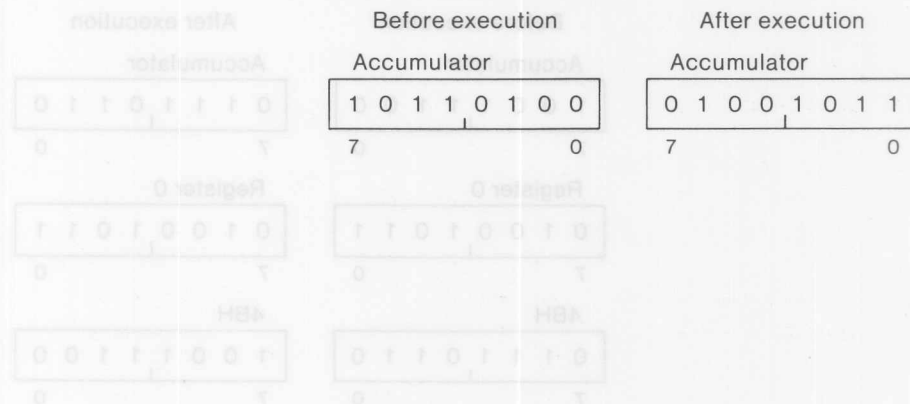
Description : The contents of the four higher order bits (4 thru 7) of the accumulator are exchanged with the contents of the four lower order bits (0 thru 3).

Example SWAP A

Instruction code : 

7	0
1	0
1	0
0	0
0	1
0	0
0	0

 Byte 1





**102. XCH A, @Rr (Exchange indirect address with accumulator)**

Instruction code : 

7							0
1	1	0	0	0	1	1	r

 Byte 1

Operation	$: (A) \rightleftharpoons ((Rr)) \quad r = 0 \text{ or } 1$
-----------	---

Number of bytes : 1

Number of cycles : 1

Flags	C	AC	F0	RS1	RS0	OV	F1	P
(PSW)								●

**Description** : The accumulator contents are exchanged with the data memory location contents addressed by the register *r* contents, and the flag is updated.

Example    XCH A, @R0

Instruction code

7	0
1	1
0	0
0	1
1	1
0	

Byte 1

Before execution

After execution

Accumulator

Accumulator

1 0 0 1 1 1 0 0

7 0

0 1 1 1 0 1 1 0  
7 0

Register 0

Register 0

0 1 0 0 1 0 1 1

0 1 0 0 1 0 1 1

4BH

4BH

0	1	1	1	0	1	1	0
7							0

$$\begin{array}{ccccccc} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 7 & & & & & & & 0 \end{array}$$

103. XCH A, Rr (Exchange register with accumulator)

Instruction code : 

7	0
1	1
0	0
1	r <sub>2</sub>
r <sub>1</sub>	r <sub>0</sub>

 Byte 1

Operation : (A)  $\rightleftharpoons$  (Rr) r = 0 ~ 7

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

							•
--	--	--	--	--	--	--	---

Description : The accumulator contents are exchanged with the register r contents, and the flag is updated.

Example XCH A, R5

Instruction code : 

7	0
1	1
0	0
1	1
0	1
1	0
1	0
1	1

 Byte 1

Before execution

Accumulator

0	1	1	1	0	0	1	0
7							0

Register 5

1	0	1	0	0	0	0	1
7							0

After execution

Accumulator

1	0	1	0	0	0	0	1
7							0

Register 5

0	1	1	1	0	0	1	0
7							0

104. XCH A, data address (Exchange memory with accumulator)

Instruction code : 

7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	1

 Byte 1

Data address : 

7	6	5	4	3	2	1	0
a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (A)  $\leftrightarrow$  (data address)

Number of bytes : 2

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

							●
--	--	--	--	--	--	--	---

Description : The accumulator contents are exchanged with the specified data address contents, and the flag is updated.

Example XCH A, 7AH

Instruction code : 

7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	1

 Byte 1

7	6	5	4	3	2	1	0
0	1	1	1	1	0	1	0

 Byte 2

Before execution

Accumulator

1	0	1	1	1	1	0	1
7							0

7AH

1	1	0	1	1	1	0	0
7							0

After execution

Accumulator

1	1	0	1	1	1	0	0
7							0

7AH

1	0	1	1	1	1	0	1
7							0

105. XCHD A, @Rr (Exchange low nibbles of indirect address with accumulator)

Instruction code : 

7	0
1	1
0	1
1	0
1	1
r	r

 Byte 1

Operation :  $(A_{0-3}) \leftrightarrow ((Rr_{0-3}))$   $r = 0 \text{ or } 1$

Number of bytes : 1

Number of cycles : 1

Flags (PSW) : 

C	AC	F0	RS1	RS0	OV	F1	P
							●

Description : The lower order bits (0 thru 3) of the accumulator contents are exchanged with the contents of the lower order bits (0 thru 3) of the data memory location addressed by register r contents. The flag is updated.

Example XCHD A, @R0

Instruction code : 

7	0
1	1
0	1
1	0
1	1
1	0

 Byte 1

Before execution

After execution

Accumulator

Accumulator

1	1	1	1	0	1	1	0
7						0	

1	1	1	1	1	1	0	1
7						0	

Register 0

Register 0

0	1	1	0	0	0	0	0
7						0	

0	1	1	0	0	0	0	0
7						0	

60H

60H

0	0	0	0	1	1	0	1
7						0	

0	0	0	0	0	1	1	0
7						0	

7

106. XRL A, #data (Logical exclusive OR immediate data to accumulator)

Instruction code : 

7	0
0	1
1	0
0	0
1	0
0	0

 Byte 1

#Data : 

7	0
I <sub>7</sub>	I <sub>6</sub>
I <sub>5</sub>	I <sub>4</sub>
I <sub>3</sub>	I <sub>2</sub>
I <sub>1</sub>	I <sub>0</sub>

 Byte 2

Operation : (A) ← (A) XOR #data

Number of bytes : 2

Number of cycles : 1

Flags : 

C	AC	F0	RS1	RS0	OV	F1	P
							●

  
(PSW)

Description : The exclusive OR operation is executed between an immediate data value and the accumulator contents. The result is placed in the accumulator, and the flag is updated.

Example XRL A, #15H

Instruction code : 

0	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---

 Byte 1

7	0
0	0
0	1
0	1
0	1
0	1

 Byte 2

Before execution

After execution

Accumulator

Accumulator

0	1	0	0	1	0	1	0
---	---	---	---	---	---	---	---

0	1	0	1	1	1	1	1
---	---	---	---	---	---	---	---

107. XRL A, @Rr (Logical exclusive OR indirect address to accumulator)

Instruction code : 

7	0
0	1
1	0
0	0
1	1
r	0

 Byte 1

Operation : (A) ← (A) XOR ((Rr)) r = 0 or 1

Number of bytes : 1

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

								●
--	--	--	--	--	--	--	--	---

Description : The exclusive OR operation is executed between the accumulator contents and the data memory location contents addressed by the register r contents. The result is placed in the accumulator, and the flag is updated.

Example XRL A, @R1

Instruction code : 

7	0
0	1
1	0
0	0
1	1
1	1

 Byte 1

Before execution

After execution

Accumulator

Accumulator

0	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

1	1	0	1	1	0	0	0
---	---	---	---	---	---	---	---

7 0

7 0

Register 1

Register 1

0	0	1	1	0	1	1	0
---	---	---	---	---	---	---	---

0	0	1	1	0	1	1	0
---	---	---	---	---	---	---	---

7 0

7 0

36H

36H

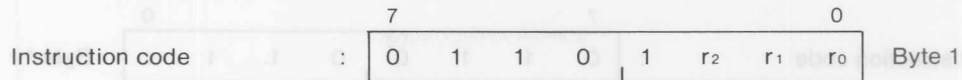
1	0	0	1	0	0	0	1
---	---	---	---	---	---	---	---

1	0	0	1	0	0	0	1
---	---	---	---	---	---	---	---

7 0

7 0

**108. XRL A, Rr (Logical exclusive OR register to accumulator)**



Operation :  $(A) \leftarrow (A) \text{ XOR } (Rr) \quad r = 0 \sim 7$

Number of bytes : 1

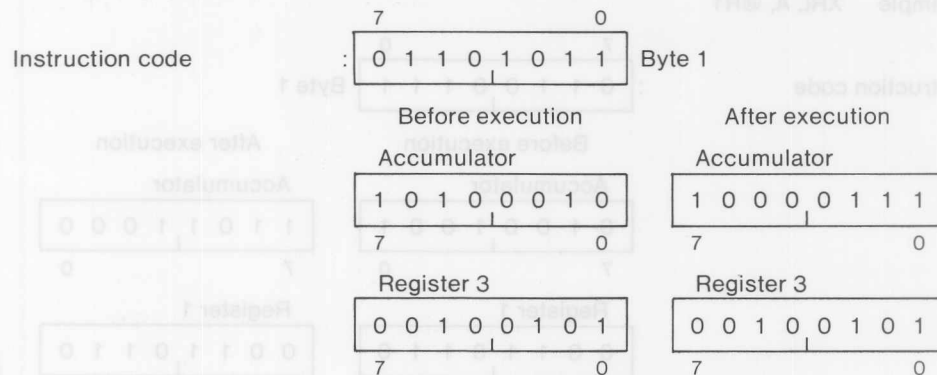
Number of cycles : 1

Flags	C	AC	F0	RS1	RS0	OV	F1	P
9	13	VO	08	1	0	0	1	0

[illegible]

**Description** : The exclusive OR between the accumulator contents and the register r contents is determined. The result is stored in the accumulator and the flag is also updated.

Example XRL A, R3



109. XRL A, data address (Logical exclusive OR memory to accumulator)

Instruction code : 

7	0
0	1
1	0
0	0
1	0
0	1

 Byte 1

Data address : 

7	0
a <sub>7</sub>	a <sub>6</sub>
a <sub>5</sub>	a <sub>4</sub>
a <sub>3</sub>	a <sub>2</sub>
a <sub>1</sub>	a <sub>0</sub>

 Byte 2

Operation : (A) ← (A) XOR (data address)

Number of bytes : 2

Number of cycles : 1

Flags : C AC F0 RS1 RS0 OV F1 P  
(PSW) 

							●
--	--	--	--	--	--	--	---

Description : The exclusive OR between the accumulator contents and the specified data address contents is determined. The result is stored in the accumulator and the flag is updated.

Example XRL A, 70H

Instruction code : 

7	0
0	1
1	0
0	0
1	0
0	1

 Byte 1

7	0
0	1
1	1
0	0
0	0
0	0

 Byte 2

Before execution

Accumulator

1	1	1	1	0	1	1	0
---	---	---	---	---	---	---	---

7 0

70H

0	0	1	0	1	1	1	1
---	---	---	---	---	---	---	---

7 0

After execution

Accumulator

1	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---

7 0

70H

0	0	1	0	1	1	1	1
---	---	---	---	---	---	---	---

7 0



110. XRL data address, #data (Logical exclusive OR immediate data to memory)

Instruction code	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	7	6	5	4	3	2	1	0	0	1	1	0	0	0	1	1	Byte 1
7	6	5	4	3	2	1	0												
0	1	1	0	0	0	1	1												
Data address	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>a<sub>7</sub></td><td>a<sub>6</sub></td><td>a<sub>5</sub></td><td>a<sub>4</sub></td><td>a<sub>3</sub></td><td>a<sub>2</sub></td><td>a<sub>1</sub></td><td>a<sub>0</sub></td></tr></table>	7	6	5	4	3	2	1	0	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2
7	6	5	4	3	2	1	0												
a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>												
#data	:	<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>l<sub>7</sub></td><td>l<sub>6</sub></td><td>l<sub>5</sub></td><td>l<sub>4</sub></td><td>l<sub>3</sub></td><td>l<sub>2</sub></td><td>l<sub>1</sub></td><td>l<sub>0</sub></td></tr></table>	7	6	5	4	3	2	1	0	l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>	Byte 3
7	6	5	4	3	2	1	0												
l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>												
Operation	:	(data address) ← (data address) XOR #data																	
Number of bytes	:	3																	
Number of cycles	:	2																	
Flags	:	C	AC	F0	RS1	RS0	OV	F1	P										
(PSW)	:																		

Description : The exclusive OR between an immediate data value and the specified data address contents is determined. The result is stored in the specified data address.

Example XRL ACC, #5AH

Instruction code	:	<div> <div>7</div> <div>0</div> <div>0</div><div>1</div><div>1</div><div>0</div> <div>0</div><div>0</div><div>1</div><div>1</div> </div>	Byte 1
		<div> <div>7</div> <div>0</div> <div>1</div><div>1</div><div>1</div><div>0</div> <div>0</div><div>0</div><div>0</div><div>0</div> </div>	Byte 2
		<div> <div>7</div> <div>0</div> <div>0</div><div>1</div><div>0</div><div>1</div> <div>1</div><div>0</div><div>1</div><div>0</div> </div>	Byte 3
Before execution			
Accumulator			
		<div> <div>7</div> <div>0</div> <div>1</div><div>1</div><div>1</div><div>1</div> <div>1</div><div>0</div><div>1</div><div>0</div> </div>	
After execution			
Accumulator			
		<div> <div>7</div> <div>0</div> <div>1</div><div>0</div><div>1</div><div>0</div> <div>0</div><div>0</div><div>0</div><div>0</div> </div>	

## DESCRIPTION OF INSTRUCTIONS

### 111. XRL data address, A (Logical exclusive OR accumulator to memory)

Instruction code	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span style="margin-right: 5px;">7</span> <span style="margin-left: 5px;">0</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span>0</span><span>1</span><span>1</span><span>0</span><span>0</span><span>0</span><span>1</span><span>0</span> </div> <span style="margin-left: 10px;">Byte 1</span>
Data address	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span style="margin-right: 5px;">7</span> <span style="margin-left: 5px;">0</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span>a<sub>7</sub></span><span>a<sub>6</sub></span><span>a<sub>5</sub></span><span>a<sub>4</sub></span><span>a<sub>3</sub></span><span>a<sub>2</sub></span><span>a<sub>1</sub></span><span>a<sub>0</sub></span> </div> <span style="margin-left: 10px;">Byte 2</span>
Operation	:	(data address) ← (data address) XOR (A)
Number of bytes	:	2
Number of cycles	:	1
Flags (PSW)	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span>C</span><span>AC</span><span>F0</span><span>RS1</span><span>RS0</span><span>OV</span><span>F1</span><span>P</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span></span><span></span><span></span><span></span><span></span><span></span><span></span><span></span> </div>
Description	:	The exclusive OR between the accumulator and the specified data address contents is determined. The result is stored in the specified data address.

Example XRL 20H, A

Instruction code	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span style="margin-right: 5px;">7</span> <span style="margin-left: 5px;">0</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span>0</span><span>1</span><span>1</span><span>0</span><span>0</span><span>0</span><span>1</span><span>0</span> </div> <span style="margin-left: 10px;">Byte 1</span>
	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span style="margin-right: 5px;">7</span> <span style="margin-left: 5px;">0</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span>0</span><span>0</span><span>1</span><span>0</span><span>0</span><span>0</span><span>0</span><span>0</span> </div> <span style="margin-left: 10px;">Byte 2</span>
Before execution		
Accumulator		
	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span style="margin-right: 5px;">7</span> <span style="margin-left: 5px;">0</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span>0</span><span>1</span><span>1</span><span>1</span><span>0</span><span>1</span><span>0</span><span>1</span> </div>
	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span style="margin-right: 5px;">7</span> <span style="margin-left: 5px;">0</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span>1</span><span>1</span><span>0</span><span>1</span><span>0</span><span>0</span><span>1</span><span>1</span> </div>
20H		
After execution		
Accumulator		
	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span style="margin-right: 5px;">7</span> <span style="margin-left: 5px;">0</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span>0</span><span>1</span><span>1</span><span>1</span><span>0</span><span>1</span><span>0</span><span>1</span> </div>
	:	<div style="display: flex; align-items: center; justify-content: space-between;"> <span style="margin-right: 5px;">7</span> <span style="margin-left: 5px;">0</span> </div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center; justify-content: space-between;"> <span>1</span><span>0</span><span>1</span><span>0</span><span>0</span><span>1</span><span>1</span><span>0</span> </div>
20H		



## **8. MSM80C51 VS PIGGY BACK**

---

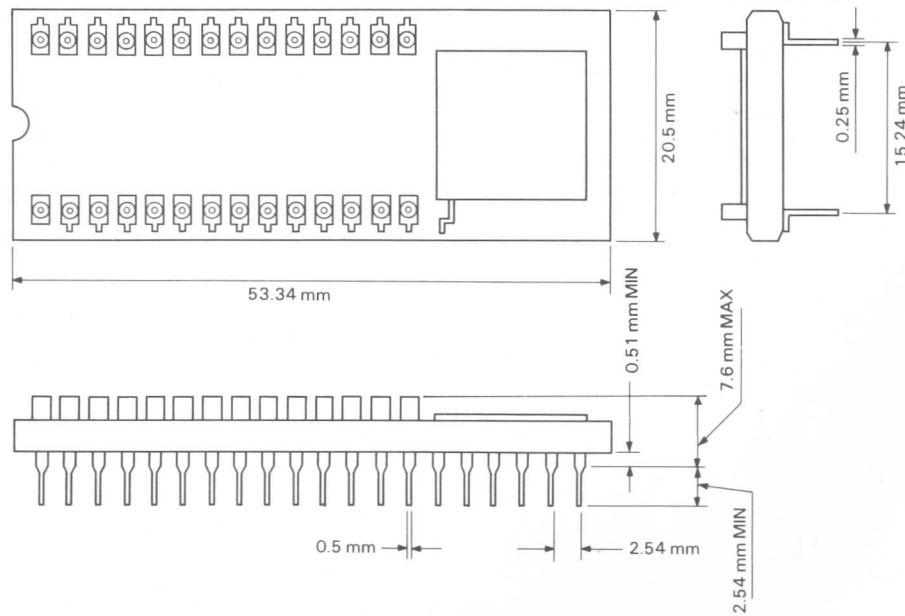
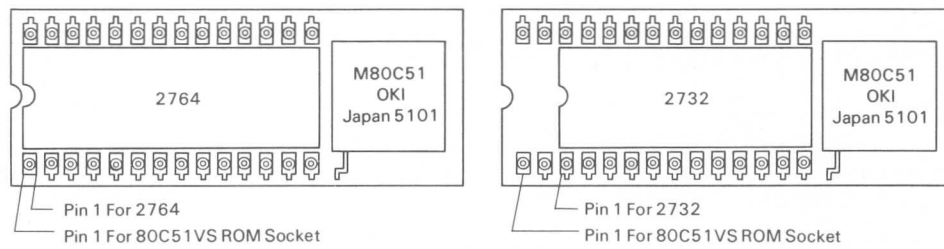
## 8. MSM80C51VS PIGGY BACK

The MSM80C51VS is a 80C51 whose built-in ROM is replaced by external EPROM using the piggy back method. The program developed by EASE 80C51 programming support tool is written into EPROM.

Since timings and electrical characteristics of the MSM80C51VS except for the storage temperature ( $-55 \sim +125^{\circ}\text{C}$ ) and the operating voltage ( $4.5 \sim 5.5\text{V}$ ) are the same as that for the MSM80C51, evaluation of programs can be done by installing the MSM80C51VS on the user's equipment directly. Therefore using the MSM80C51VS can eliminate evaluation of engineering sample and shorten TAT for developing products.

Either 2764 (64K) or 2732 (32K) may be used as EPROM.

Pin layout and external dimensions of package are shown in Figure 8-1.



Note: All The Dimensions  
Are Typical Value.

40 Lead Package (For Piggy-Back)

Figure 8-1 Pin layout and package dimensions of MSM80C51VS